VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

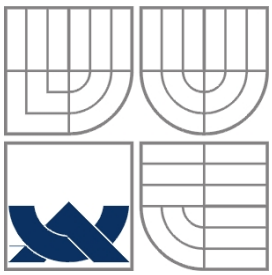
DOTAZOVÁNÍ NAD ČASOPROSTOROVÝMI DATY POHYBUJÍCÍCH SE OBJEKTŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

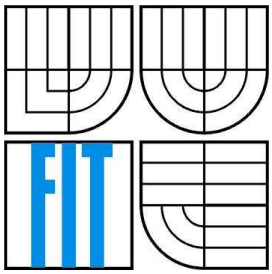
AUTOR PRÁCE
AUTHOR

Bc. ONDŘEJ DVOŘÁČEK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DOTAZOVÁNÍ NAD ČASOPROSTOROVÝMI DATY POHYBUJÍCÍCH SE OBJEKTŮ

QUERYING SPATIO-TEMPORAL DATA OF MOVING OBJECTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ONDŘEJ DVOŘÁČEK

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. JAROSLAV ZENDULKA, CSc.

BRNO 2009

Abstrakt

Diplomová práce je věnována studiu možností, jakými lze reprezentovat data pohybujících se objektů a jak je možné se nad těmito časoprostorovými daty dotazovat. Dále jsou zde shrnuty výsledky diplomové práce pana Ing. Jaroslava Vališe, ze kterých se při řešení této diplomové práce mělo vycházet. Na základě získaného teoretického základu, prezentovaného na začátku práce, však byla navržena a implementována zcela nová podpora pro uložení časoprostorových dat a pro obecné dotazování se nad těmito daty. Její konkrétní využití je pak dále demonstrováno v ukázkové aplikaci, která mimo jiné toto řešení využívá k implementaci svých složitějších doménově specifických databázových operací. Na závěr jsou uvedeny hlavní směry dalšího vývoje navrženého databázového rozšíření a zasazení výsledků této práce do kontextu pokračujícího projektu, disertační práce na téma „Databáze pohybujících se objektů“.

Klíčová slova

Databáze, dotazování, pohybující se objekty, časoprostorová data, Oracle, Oracle Spatial, STD.

Abstract

This master’s thesis is devoted to the studies of possibilities, which can be used for representation of moving objects data and for querying such spatio-temporal data. It also shows results of the master’s thesis created by Ing. Jaroslav Vališ, that should be used for the solution of this master’s thesis. But based on the theoretical grounds defined at the beginning of this work was designed and implemented new database extension for saving and querying spatio-temporal data. Special usage of this extension is demonstrated in an example application. This application uses the database extension for the implementation of its own database functions that are domain specific. At the conclusion, there are presented ways of the farther development of this database extension and the results of this master’s thesis are there set into the context of the following project, doctoral thesis “Moving objects database”.

Keywords

Database, querying, moving objects, spatio-temporal data, Oracle, Oracle Spatial, STD.

Citace

Dvořáček Ondřej: Dotazování nad časoprostorovými daty pohybujících se objektů. Brno, 2009, diplomová práce, FIT VUT v Brně.

Dotazování nad časoprostorovými daty pohybujících se objektů

Prohlášení

Prohlašuji, že jsem pod vedením doc. Ing. Jaroslava Zendulky, CSc., vypracoval tuto diplomovou práci samostatně a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bc. Ondřej Dvořáček
26. 5. 2009

Poděkování

Chtěl bych poděkovat panu doc. Ing. Jaroslavu Zendulkovi, CSc., za jeho cenné rady a čas, který mi během tvorby této práce věnoval.

Obsah

1 Úvod

Databáze pohybujících se objektů se čím dál víc dostávají do popředí díky narůstajícímu počtu využití v různých senzorových nebo kamerových systémech. Jejím cílem je uchovávat informace o průběhu změny polohy pozorovaného objektu v dimenzích prostoru a času, tedy takzvaná časoprostorová data. Tato data však nemusí sloužit pouze pro uchování dat pohybujících se objektů, ale jejich pole použití je daleko rozsáhlejší (např. evidence průběhu změny teplot, a mnoho dalších veličin měnících se s časem). Bohužel v současné době zatím neexistuje žádný databázový systém, který by nabízel speciální podporu pro uchování těchto dat, případně pak i možnost se nad takovými daty dotazovat. Řešení zmíněného problému je stále předmětem dalšího, poměrně intenzivního, vývoje.

Z názvu diplomové práce je patrné, že jeho smyslem bude snaha o vytvoření databázové podpory, která by umožnila dotazování nad časoprostorovými daty pohybujících se objektů. Prostředkem k řešení zadaných cílů mělo být využití časoprostorové databáze navržené a implementované v diplomové práci pana Ing. Jaroslava Vališe nazvané „Databáze pohybujících se objektů“ (viz [2]). Jak se ale později blíže přesvědčíme, je jeho řešení pro obecné dotazování nad časoprostorovými daty nevhodné a zatížené doménou svého použití, konkrétně oblastí dohledových či senzorových systémů.

Nejdříve se ve druhé kapitole podíváme na to, jakým způsobem je vůbec možné časoprostorová data reprezentovat. Pokusíme se zde zavést formálně systém typů a především operace nad těmito typy, které budou později implementovány jako základní podpora pro dotazování nad časoprostorovými daty pohybujících se objektů.

Dalším krokem bude získání informací o podpoře pro uchování časoprostorových dat, respektive možnosti reprezentace typů a operací, zavedených v předchozí kapitole, v databázovém systému Oracle 11g.

Kapitola čtvrtá je věnována detailnímu popisu původní databáze pohybujících se objektů navržené panem Ing. Jaroslavem Vališem. Podrobně jsou zde rozebrány jednotlivé části navržené architektury a implementované operace, z jejichž prostudování je patrné využití základních operací k vytvoření konkrétní implementace v doméně časoprostorových dat.

V další kapitole bude představena hlavní část této diplomové práce, a to ucelený souhrn datových typů a balíčků operací odpovídajících jejich formálnímu zavedení v kapitole druhé. Vytvořením tohoto časoprostorového databázového rozšíření, implementovaného pro databázový systém Oracle 11g, byl položen silný základ pro další studium této problematiky, jehož využití sahá mnohem dále, než pouze k aplikační doméně dohledových či senzorových systémů.

Jako ukázka použití vytvořeného časoprostorového databázového rozšíření, zatím pracovně nazvaného STD (název odvozen od počátečních písmen anglického překladu spojení časoprostorová

databáze – „*Spatio-Temporal Database*“), byla implementována jednoduchá webová aplikace využívající právě tohoto databázového rozšíření k implementaci svých složitějších a doménově specifických databázových funkcí.

Porovnání původní databáze pohybujících se objektů s databázovým rozšířením STD a dále také prezentování námětů na další vývoj tohoto rozšíření je věnována kapitola sedmá. I přes rozsáhlou implementovanou podporu pro dotazování nad časoprostorovými daty pohybujících se objektů, je toto řešení pouze malou částí pokrývající celou doménu časoprostorových dat. Slouží tak především jako studijní část pro další práci na tomto tématu při mém doktorském studiu a uvádí hlavní směry, kterými by se další vývoj v této oblasti měl ubírat.

V závěru práce jsou pak shrnuty dosažené výsledky, které jsou dále zasazeny do kontextu řešení daleko rozsáhlejšího pokračujícího projektu, a to disertační práce na téma „Databáze pohybujících se objektů“.

2 Pohybující se objekty a jejich reprezentace

K řešení problému, kterým se tato diplomová práce zabývá, tedy dotazování nad daty pohybujících se objektů, je nejdříve nutné stanovit množinu základních operací, jež budou základem vzniklého databázového rozšíření využitelného pro implementaci složitějších a specifických časoprostorových dotazů. Takové základní operace je samozřejmě nutné definovat nad formálně zavedeným systémem datových typů, pomocí nichž bude možné data pohybujících se objektů (časoprostorová data) reprezentovat.

Následující podkapitoly jsou tedy věnovány nejdříve popisu systému datových typů, který byl v našem případě zvolen pro reprezentaci časoprostorových dat, a dále zavedení formálně specifikovaných operací, které lze nad těmito datovými typy provádět. Obě tyto části shrnují výsledky práce Ralfa Hartmuta Gütinga [1], od níž se také odvíjel vývoj databáze pohybujících se objektů navržené panem Ing. Jaroslavem Vališem (viz [2]).

2.1 Systém datových typů časoprostorových dat

Systém datových typů, které jsou potřeba k reprezentaci časoprostorových dat, můžeme rozdělit do několika tříd datových typů. Tyto třídy postupně rozebereme, zavedeme jejich názvy a jednotlivé datové typy, které do těchto tříd patří.

2.1.1 Základní datové typy

Základní datové typy řadíme do třídy datových typů označené jako BASE. Do této třídy potom patří typy jako *int*, *real*, *string* nebo *bool*, tak jak jsou běžně známy, ovšem rozšířené o nedefinovanou hodnotu. Hodnoty uvedených typů náleží do jednorozměrného prostoru.

2.1.2 Prostorové datové typy

Do třídy datových typů označené jako SPATIAL řadíme prostorové typy bod, množina bodů, křivka a region (jejich grafická reprezentace viz obrázek 2.1). Názvy konstruktorů těchto typů jsou pojmenovány podle anglických názvů typů, a to: *point*, *points*, *line*, *region*. Na rozdíl od základních typů mapují své hodnoty do dvourozměrného prostoru.

Obrázek 2.1: Prostorové datové typy (převzato z [1]).

2.1.3 Časové datové typy

Jediným časovým typem, který řadíme do třídy typů nazvané TIME, je časový okamžik, jehož množina hodnot odpovídá množině reálných čísel spolu s nedefinovanou hodnotou. Stejně jako v předchozím případě prostorových datových typů odpovídá název konstruktoru tohoto typu jeho anglickému překladu, *instant*. Časový okamžik považujeme za jednorozměrnou lineárně spojitou veličinu.

2.1.4 Temporální datové typy

Třída datových typů TEMPORAL, neboli třída temporálních datových typů, je množina typů odvozených pomocí konstruktorů *moving* a *intime*.

Typovým konstruktorem *moving* mapujeme k hodnotám datových typů z tříd BASE a SPATIAL jejich časový vývoj. Vznikají tak nové datové typy *mint*, *mreal*, *mstring*, *mbool*, *mpoint*, *mpoints*, *mline* a *mregion*. Tyto typy jsou potom reprezentovány buď funkcemi času, nebo konečnou množinou párů (okamžik, hodnota).

Další datové typy sdružující pouze jednu hodnotu časového okamžiku (typ *instant*) s hodnotou datového typu náležejícího rovněž do třídy BASE nebo SPATIAL je možné získat typovým konstruktorem třídy TEMPORAL nazvaným *intime*. Získáme tak jediný pár (okamžik, hodnota).

V závislosti na tom, z jaké třídy datových typů (BASE nebo SPATIAL) jsou nové temporální typy odvozeny, náleží hodnoty těchto typů do jedno, nebo dvourozměrného prostoru.

2.1.5 Rozsahové datové typy

Rozsahové datové typy řadíme do třídy datových typů nazvané RANGE. Jedná se o typy odvozené ze základních datových typů (třída BASE) pomocí typového konstruktoru *range* a z časového typu *instant* (třída TIME), pro který vznikl speciální konstruktor nazvaný *periods* (odpovídá *range(instant)*). Pomocí zmíněných konstruktorů tak můžeme vytvářet typy reprezentující množiny intervalů nad časovou doménou, doménou reálných nebo celých čísel, atd.

2.2 Operace nad časoprostorovými daty

Při popisu jednotlivých operací nad datovými typy zavedenými v předchozí kapitole budeme používat následující notaci:

- Symbol π – typ, jehož proměnné nabývají pouze jedné hodnoty.
- Symbol σ – typ, jehož proměnné nabývají množiny hodnot.
- Symbol α, β – typ, jehož proměnné nabývají hodnot základních a prostorových typů.
- Symboly *k*, *l* – označují jednotlivé hodnoty proměnných typu π.
- Symboly *K*, *L* – označují jednotlivé hodnoty proměnných typu σ.
- Symbol ∂K – označuje hranici bodů množiny *K*.
- Symbol $K°$ – označuje vnitřek množiny *K*.
- Symbol $\rho(K)$ – označuje topologii, na níž byla provedena uzávěrová operace.
- Označení *[nD]* – omezuje takto označené operace do *n*-dimenziálního prostoru.
- Velká písmena abecedy – označují název proměnné, která je datového typu uvedeného před ní, např. *line L1 – L1* je proměnná typu *line*.

2.2.1 Operace nad netemporálními datovými typy

Operace nad netemporálními datovými typy můžeme rozdělit do několika kategorií podle toho, o jaký druh operací se jedná.

První kategorií jsou predikátové operace. Mohou být buď unární, nebo binární a jedná se o následující operace:

- *bool* ***is_empty****(π k); bool* ***is_empty****(σ K)*
  Vrací *true* v případě, že je argument *k* nedefinovaný, nebo že je množina *K* prázdná.
- $k = l; k \neq l; K = L; K \neq L$
  Vrací *true* pokud si jsou, respektive nejsou, argumenty *k* a *l* nebo množiny *K* a *L* rovny.
- *bool* ***intersects****(σ K, σ L)*
  Vrací *true*, pokud mají množiny *K* a *L* neprázdný průnik.
- *bool* ***inside****(σ K, σ L)*; *bool* ***inside****(π k, σ L)*
  Vrací *true*, pokud je množina *K* podmnožinou *L* nebo pokud je argument *k* obsažen v množině *L*.
- $k < l, k \leq l, k \geq l, k > l$ *[1D]*
  Vrací *true*, pokud je v jednodimenzionálním prostoru argument *k* menší než *l* (další analogicky).
- *bool* ***before****(σ K, σ L)[1D]; bool* ***before****(π k, σ L)[1D]; bool* ***before****(σ K, π l)[1D]*
  Vrací *true*, pokud v jednodimenzionálním prostoru je každý prvek množiny *K* před všemi prvky množiny *L*, nebo pokud je argument *k* před všemi prvky množiny *L*, nebo pokud všechny prvky množiny *L* jsou před argumentem *l*.

- *bool* ***touches****(σ K, σ L)*

  Vrací *true*, pokud se hranice množin *K* a *L* dotýkají, tedy průnik *∂K* a *∂L* je neprázdná množina.
- *bool* ***attached****(σ K, σ L)*

  Vrací *true*, pokud jsou množiny *K* a *L* propojené, tedy průnik *∂K* a *L°* je neprázdná množina.
- *bool* ***overlaps****(σ K, σ L)*

  Vrací *true*, pokud se množiny *K* a *L* překrývají, tedy průnik *K°* a *L°* je neprázdná množina.
- *bool* ***on_border****(π k, σ L)*

  Vrací *true*, pokud argument *k* je prvkem hranice množiny *L*, tedy množiny *∂L*.
- *bool* ***in_interior****(π k, σ L)*

  Vrací *true*, pokud se argument *k* nachází uvnitř množiny *L*, tedy množiny *L°*.

Další kategorií jsou operace množinové. Jejich výčet je uveden níže:

- *π* ***intersection****(π k, π l); π* ***intersection****(π k, σ L); σ* ***intersection****(σ K, σ L)*

  Vrací *k*, pokud se *k* rovná *l* nebo *k*, pokud je *k* obsaženo v množině *L*, nebo průnik množin *K* a *L*. V ostatních případech vrací nedefinovanou hodnotu.
- *π* ***minus****(π k, π l); π* ***minus****(π k, σ L); σ* ***minus****(σ K, π l); σ* ***minus****(σ K, σ L)*

  Pro argumenty *k* a *l* vrací nedefinovanou hodnotu, pokud se *k* rovná *l*, jinak vrací *k*.

  Pro argumenty *k* a *L* vrací nedefinovanou hodnotu, je-li *k* prvkem množiny *L*, jinak vrací *k*.

  V ostatních případech, jestliže je *K* (respektive *K* a *L*) množina dvoudimenzionálních prvků, vrací rozdíl množiny *K* a jednoprvkové množiny obsahující *l* (respektive množin *K* a *L*).
- *π* ***union****(π k, σ L); σ* ***union****(σ K, π l); σ* ***union****(σ K, σ L)*

  Pokud jsou prvky množiny *L* jednodimenzionální, nebo jsou typu *points*, pak vrací množinu *L* (respektive *K*) rozšířenou o prvek *k* (respektive *l*), jinak vrací množinu *L* (respektive *K*).

  V případě argumentů *K* a *L* vrací sjednocení těchto dvou množin.
- *points* ***crossings****(line L1, line L2)*

  Vrací množinu průsečíků dvou křivek *L1* a *L2*.
- *points* ***touch_points****(region R, line L); points* ***touch_points****(line L, region R);*
  points ***touch_points****(region R1, region R2)*

  Vrací množinu průsečíků regionu *R* s křivkou *L* nebo hraničních kontur regionů *R1* a *R2*.

Operace, které redukují množinu hodnot do jedné hodnoty, patří do kategorie agregačních operací, jejichž popis je v následujícím textu:

- *π* ***min****(σ K); π* ***max****(σ K)[1D]*

  Vrací minimální, respektive maximální hodnotu množiny *K* v jednodimenzionálním prostoru.
- *π* ***avg****(σ K)*

  Vrací průměrnou hodnotu množiny *K*.

- *π* ***sum****(σ K)[1D]*

  Vrací součet všech prvků množiny *K* v jednodimenzionálním prostoru.
- *π* ***count****(σ K)*

  Vrací počet prvků množiny *K*.

Poslední kategorií jsou operace vzdálenosti a směru. Jedná se o operace:

- *π* ***distance****(π k, π l); π* ***distance****(π k, σ L); π* ***distance****(σ K, σ L)*

  Vrací minimální vzdálenost nejbližší dvojice bodů z prvního a druhého argumentu.
- *π* ***direction****(π k, π l)*

  Vrací úhel přímky tvořené body *k* a *l* s osou *x*, pokud jsou body totožné, vrací nedefinovanou hodnotu.

2.2.2 Operace nad temporálními datovými typy

Stejně jako i u netemporálních datových typů můžeme operace nad temporálními datovými typy rozdělit do kategorií podle typů těchto operací.

Do první kategorie operací řadíme operace, které slouží k projekci do domény a rozsahu:

- *periods* ***deftime****(moving(α) M)*

  Vrací čas, po který je hodnota argumentu *M* definována.
- *range(α)* ***rangevalues****(moving(α) M)[1D]*

  Vrací pro typy z jednodimenzionálního prostoru hodnoty argumentu *M* přiřazené během času jako interval těchto hodnot.
- *points* ***locations****(moving(point) M); points* ***locations****(moving(points) M)*

  Vrací projekci argumentu *M* do množiny bodů v prostoru.
- *line* ***trajectory****(moving(point) M); line* ***trajectory****(moving(points) M)*

  Vrací projekci argumentu *M* do křivky v prostoru.
- *region* ***traversed****(moving(line) M); region* ***traversed****(moving(region) M)*

  Vrací pro argument *M* uzavřený region, ve kterém se pohyboval.
- *instant* ***inst****(intime(α) I)*

  Vrací projekci argumentu *I* do domény časového typu *instant*.
- *α* ***val****(intime(α) I)*

  Vrací projekci argumentu *I* do domény typu *α*.

Další skupinu operací tvoří operace selekce pohybujících se objektů v čase:

- *intime(α)* ***atinstant****(moving(α) M, instant T)*

  Vrací pár (okamžik, hodnota), jako výsledek omezení argumentu *M* časovým okamžikem *T*.
- *moving(α)* ***atperiods****(moving(α) M, periods P)*

  Vrací množinu párů (okamžik, hodnota), jako výsledek omezení argumentu *M* intervalem *P*.

- *intime(α)* ***initial****(moving(α) M)*
  Vrací počáteční pár (okamžik, hodnota) argumentu *M*.
- *intime(α)* ***final****(moving(α) M)*
  Vrací poslední pár (okamžik, hodnota) argumentu *M*.
- *bool* ***present****(moving(α) M, instant T); bool* ***present****(moving(α) M, periods P)*
  Vrací *true*, pokud argument *M* existoval v čase *T* nebo v časovém intervalu *P*.
- *moving(α)* ***at****(moving(α) M, α A)[1D]; moving(α)* ***at****(moving(α) M, range(α) R)[1D];*
  mpoint ***at****(moving(α) M, point P)[2D]; moving(α)* ***at****(moving(α) M, β B)[2D]*
  Pro typy *α* z jednodimenzionálního prostoru vrací omezení argumentu *M* hodnotou argumentu *A* nebo rozsahem hodnot argumentu *R*, jehož výsledkem je množina párů (okamžik, hodnota).
  Pro typy *α* z dvoudimenzionálního prostoru vrací omezení argumentu *M* prostorovým argumentem *P* nebo *B*, jehož výsledkem je pohybující se objekt s dimenzí rovnou nejmenší dimenzi obou argumentů *M* a *P* (respektive *B*) podle uspořádání *point* < *points* < *line* < *region*.
- *moving(α)* ***atmin****(moving(α) M)[1D]*
  Vrací pro typy *α* z jednodimenzionálního prostoru minimální hodnotu argumentu *M*.
- *moving(α)* ***atmax****(moving(α) M)[1D]*
  Vrací pro typy *α* z jednodimenzionálního prostoru maximální hodnotu argumentu *M*.
- *bool* ***passes****(moving(α) M, β B)*
  Vrací *true*, pokud argument *M* nabyl hodnoty nebo hodnoty z daného rozsahu argumentu *B*.

Z vlastností proměnných temporálních typů vyplývá další kategorie operací, která slouží k určení rychlosti změny jejich hodnot. Mezi ně patří následující:

- *real* ***derivative****(mreal M)[1D]*
  Vrací rychlost změny hodnoty argumentu *M*.
- *mreal* ***speed****(mpoint M)*
  Vrací rychlosti změn hodnot argumentu *M* na základě Eukleidovské vzdálenosti jeho bodů.
- *mreal* ***turn****(mpoint M)*
  Vrací rychlosti změn hodnot argumentu *M* na základě směru mezi jeho body.
- *mpoint* ***velocity****(mpoint M)*
  Vrací rychlosti změn hodnot argumentu *M* na základě rozdílu jeho vektorů.

Další operace nad temporálními typy je možné získat přizpůsobením některých operací nad netemporálními datovými typy. Principem metody přizpůsobení je konverze proměnné typu, která nabývá jedné hodnoty nebo rozsahu hodnot, do proměnné odpovídajícího temporálního datového typu. Ta je potom tvořena párem (okamžik, hodnota) nebo množinou takových párů.

3 Podpora časoprostorových dat databázového systému Oracle 11g

Přímá podpora pro časoprostorová data se v současné době bohužel v databázových systémech zatím nevyskytuje. Využívají se tak především vlastnosti systémů zaměřených pouze na prostorová data a pouze na data časová.

Ne jinak je tomu i v případě databázového systému Oracle 11g, kde máme k dispozici zvlášť základní podporu pro uchování časových dat, a propracovanou podporu reprezentace a uchování prostorových dat v podobě databázového rozšíření Oracle Spatial.

Databázové systémy, které by zajišťovaly efektivní uložení časoprostorových dat a práci s těmito daty, jsou v současné době předmětem výzkumu. Využívá se především možnosti spojení databázových systémů zaměřených zvlášť na podporu časových a prostorových dat. Příkladem takového postupu může být vývoj databázového systému s podporou časoprostorových dat spojením databázových systémů TimeDB a Oracle Spatial (více viz [6]).

3.1 Podpora časových dat

Z hlediska podpory práce s časovými daty databázový systém Oracle nabízí datové typy DATE a TIMESTAMP. Kromě těchto dvou datových typů nabízí systém také datový typ INTERVAL včetně několika jeho variací. Pro naše potřeby vytvoření časoprostorové databáze jsou však použitelné pouze první dva výše zmíněné, proto se v dalším textu budeme věnovat výhradně těmto dvěma datovým typům.

3.1.1 Datový typ DATE

Ukládání času a data zároveň umožňuje datový typ DATE. Pomocí něho je možné do databáze uložit hodnotu století, roku, měsíce, dne, hodiny, minuty a sekundy. Databázový systém Oracle pro reprezentaci těchto hodnot používá svůj vlastní formát, který je možné změnit pomocí nastavení parametru NLS_DATE_FORMAT.

Pokud je ukládaná hodnota časového okamžiku ve standardním (případně uživatelem definovaném) formátu, lze ji zapsat jednoduše pomocí znakového řetězce. V ostatních případech je možné použít funkci TO_DATE, která se postará o převod do standardního formátu zadáním znakového řetězce spolu s formátem, ve kterém je tento řetězec zadáván (více o této funkci viz [5]).

V případě, že nebyla zadána hodnota času, je automaticky doplněna hodnota 00:00:00 nebo 12:00:00 pro 24hodinový, respektive 12hodinový formát času. V opačném případě, je-li vynecháno datum, je implicitně uložena hodnota prvního dne v aktuálním měsíci a roce.

3.1.2 Datový typ TIMESTAMP

Rozšířením datového typu DATE je datový typ TIMESTAMP, který je vhodný především pro uložení časových okamžiků s přesností zlomků sekundy. Z datového typu DATE ukládá hodnoty roku, měsíce, dne, hodiny, minuty a hodnoty sekundy se zadanou přesností zlomku sekundy. Přesnost je implicitně nastavena na hodnotu 6, přičemž ji lze změnit na jakoukoliv hodnotu celého čísla v rozmezí 0 až 9 pomocí příkazu `TIMESTAMP[presnost_zlomku_sekundy]`.

Další dvě varianty tohoto datového typu, které databázový systém Oracle nabízí, jsou datové typy TIMESTAMP WITH TIME ZONE a TIMESTAMP WITH LOCAL TIME ZONE. Oba dovolují navíc kromě uložení časových okamžiků i uchování informace o časovém pásmu (více o těchto typech v [5]).

3.2 Podpora prostorových dat – Oracle Spatial

Možnost práce s prostorovými daty v databázovém systému Oracle zajišťuje speciální rozšíření tohoto systému nazvané Oracle Spatial. Jedná se o sadu funkcí a procedur, které umožňují ukládat prostorová data, přistupovat k nim a efektivně se nad nimi dotazovat. V dalších podkapitolách jsou blíže specifikovány jednotlivé rysy této prostorové nadstavby.

3.2.1 Model dat

Pro reprezentaci geometrických objektů využívá Oracle Spatial objektově-relační databázový model. Jednotlivé objekty je díky tomuto použitému modelu databázový systém Oracle schopný reprezentovat vlastním speciálním datovým typem pro vektorová data, typem SDO_GEOMETRY.

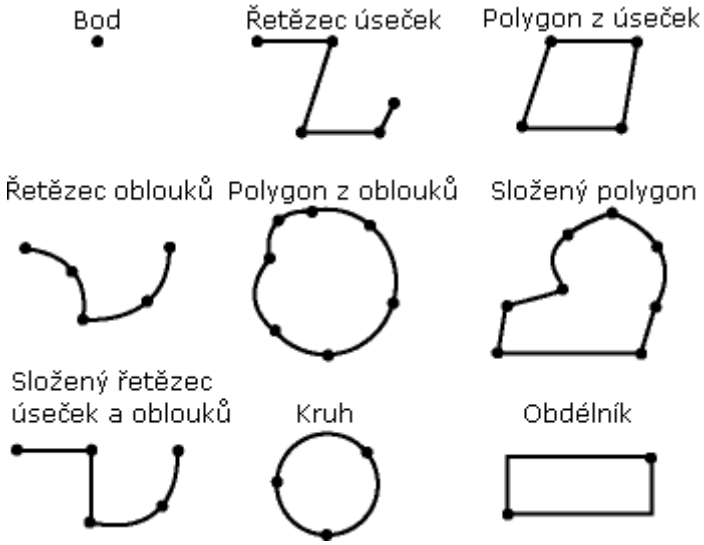
Samotný datový model prostorových dat je pak tvořen hierarchickou strukturou prostorových elementů, geometrických objektů a vrstev z těchto objektů složených.

3.2.1.1 Prostorové elementy

Prostorový element je základním stavebním blokem všech geometrických objektů. Podporovanými typy takových elementů jsou body, řetězce spojených křivek a polygony. Prostorové elementy reprezentujeme pomocí k nim přiřazených souřadnic. V případě bodu se jedná pouze o jednu souřadnici, u křivek pak o dvoje souřadnice označující začátek a konec elementu a konečně u polygonu je použito vždy páru souřadnic pro každou křivku tvořící výsledný polygon, přičemž definice těchto souřadnic probíhá proti směru hodinových ručiček pro vnější a ve směru hodinových ručiček pro vnitřní ohraničení polygonu.

3.2.1.2 Geometrické objekty

Spojením několika prostorových elementů popsaných v předchozí kapitole můžeme získat různé druhy *geometrických objektů*. Patří mezi ně například řetězec úseček nebo oblouků, polygon z úseček nebo oblouků, polygon složený z úseček a oblouků a několik dalších. Grafické znázornění možných variant ukazuje obrázek 3.1.



Obrázek 3.1: Ukázka geometrických objektů (převzato z [4]).

3.2.1.3 Vrstvy geometrických objektů

Jak už bylo výše naznačeno, *vrstvy* jsou kolekce geometrických objektů, které mají stejnou množinu atributů. Příkladem může být jedna vrstva uchovávající informace o hustotě osídlení a druhá popisující dopravní infrastrukturní síť určité oblasti. Součásti takových vrstev, tedy jednotlivé geometrické objekty a k nim vytvořené prostorové indexy, jsou standardně ukládány v klasických tabulkách databázového systému Oracle.

3.2.1.4 Souřadný systém

V průběhu popisu reprezentace jednotlivých geometrických objektů bylo zmíněno přiřazení souřadnice v určitém prostoru k jednotlivým bodům. To, jakým způsobem budou tyto souřadnice stanoveny a jaký mezi nimi bude vztah, určuje použitý *souřadný systém*. V Oracle Spatial je možné použít jeden z následujících čtyř druhů souřadných systémů:

- *Kartézský souřadný systém* – určuje souřadnice bodu na základě jeho pozice vzhledem ke zvolenému počátku po jednotlivých osách, které jsou navzájem kolmé v reprezentovaném prostoru. Pokud není souřadný systém explicitně zadán, je jako implicitní použit právě tento Kartézský souřadný systém.
- *Geodetický souřadný systém* – vyjadřuje pomocí úhlových souřadnic zeměpisnou délku a šířku. Tyto souřadnice jsou velice podobné sférickým polárním souřadnicím.

- *Projekční souřadný systém* – promítá matematickým mapováním bod na povrchu Země do roviny s Kartézským souřadným systémem.
- *Lokální souřadný systém* – odpovídá Kartézskému souřadnému systému s tím rozdílem, že souřadnice popisující polohy v tomto souřadném systému nejsou vztaženy k Zemi (používá se například v různých CAD aplikacích).

3.2.1.5 Tolerance

Při práci s prostorovými daty je často využívána *tolerance*, která určuje maximální vzdálenost mezi dvěma body v prostoru, při které ještě o těchto dvou bodech můžeme prohlásit, že jsou shodné. Specifikace hodnoty tolerance probíhá ve dvou případech, a to při definici vrstvy geometrických objektů, a dále jako jeden z parametrů při volání prostorových funkcí. Tolerance musí nabývat kladných hodnot větších než nula, přičemž platí, že čím menší číslo je zadáno, tím přesnějších výsledků je při prostorových operacích dosaženo. Zvolená hodnota tolerance je rovněž velmi závislá na konkrétním použitém souřadném systému.

3.2.2 Dotazování nad prostorovými daty

K získání výsledku dotazu nad prostorovými daty slouží v Oracle Spatial dvouvrstvý dotazovací model. Jedná se o kombinaci výsledků dvou odlišných operací označovaných jako *primární* a *sekundární* filtrační operace (princip znázorněn na obrázku 3.2).

Úkolem primární filtrační operace je rychlý výběr záznamů pro jejich zpracování pomocí sekundární filtrační operace. Porovnáním provedených geometrických odhadů snižuje výpočetní složitost vrácením množiny možných správných výsledků.

Sekundární filtrační operace už pak provádí přesný výpočet pouze na datové podmnožině, která je výstupem primární filtrační operace, nikoliv na celé datové množině. Jejím výsledkem je pak přesná odpověď na prostorový dotaz.

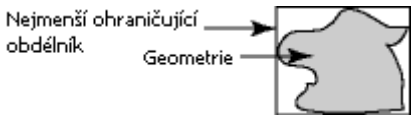
Obrázek 3.2: Model dotazování nad prostorovými daty (převzato z [4]).

K tomu, aby bylo dosaženo co nejrychlejšího zpracování dotazů, je důležitá především efektivní implementace primární filtrační operace pomocí databázových indexů na prostorová data, a tím maximální omezení množiny možných správných výsledků z celé rozsáhlé datové množiny.

3.2.3 Indexování prostorových dat

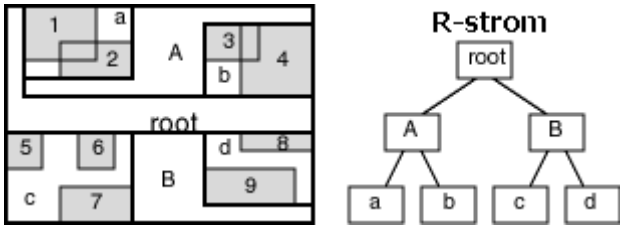
Jednou z hlavních vlastností Oracle Spatial je prostorový index. Stejně jako ostatní indexy slouží k omezení vyhledávání v datové množině, avšak mechanismus indexace je založen na prostorových kritériích, jako je průnik či inkluze.

Položky prostorového indexu jsou závislé na umístění geometrií v souřadném systému. V databázovém systému Oracle je možné vytvořit dva druhy prostorových indexů. Nejčastěji používaným je prostorový index založený na R-stromu (anglicky R-tree). Druhým možným způsobem je prostorový index vytvořený pomocí Quad-stromu (anglicky Quad-tree). Od tohoto prostorového indexu se ale v současné době upouští a je cílem jeho podporu z databázového systému Oracle zcela odstranit.



Obrázek 3.3: Nejmenší ohraničující obdélník geometrie (převzato z [4]).

Prostorový R-strom index postupně dělí prostor podle v něm obsažených geometrií a geometrie v nově získaných podprostorech se snaží obklopit nejmenším možným obdélníkem (viz obrázek 3.3). Každým dělením je tak vytvořena další vrstva v hierarchii R-stromu, přičemž listy v poslední vrstvě R-stromu už odpovídají nejmenšímu možnému dělení prostoru (viz obrázek 3.4). Takovým způsobem je možné indexovat prostorová data až o čtyřech dimenzích.



Obrázek 3.4: Hierarchický R-strom index (převzato z [4]).

3.2.4 Prostorové vztahy a filtrování

Oracle Spatial používá k určení prostorových vztahů již dříve zmíněné sekundární filtrační operace. Vztahy jsou založeny na poloze geometrií, přičemž nejvíce vztahů je založeno na topologických vlastnostech, nebo vzájemných vzdálenostech geometrií. Jako sekundární filtrační operace pro určení vzájemných vztahů mezi geometrickými objekty slouží několik následujících operací: SDO_RELATE, SDO_WITHIN_DISTANCE a SDO_NN.

SDO_RELATE operátor určuje devět možných topologických vztahů mezi body, křivkami nebo polygony. Dva objekty se tak mohou nedotýkat, dotýkat, rovnat se, obsahovat jeden druhý, překrývat jeden druhý, být jeden uvnitř druhého, atd.

SDO_WITHIN_DISTANCE určuje, zda se dva prostorové objekty navzájem nacházejí v zadané vzdálenosti od sebe.

SDO_NN vrací určitý zadaný počet prostorových objektů, které jsou nejblíže zadané prostorové geometrii.

3.2.5 Prostorové operátory, procedury a funkce

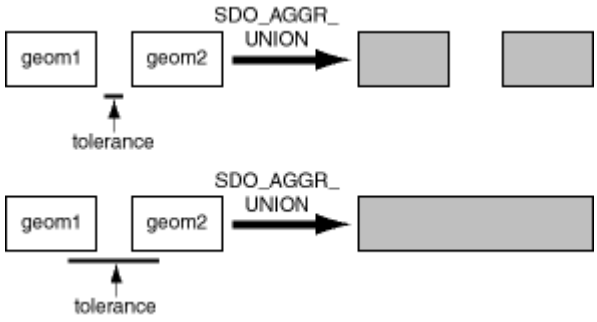
Mezi prostorové operátory patří zejména SDO_FILTER a SDO_RELATE. Používají prostorový index a díky tomu poskytují velice dobrý výkon. Mohou být použity v klauzuli WHERE ve všech prostorových dotazech.

Prostorové procedury a funkce jsou poskytovány PL/SQL balíčky podprogramů, jako například SDO_GEOM, SDO_CS, SDO_LRS. Na rozdíl od prostorových operátorů nevyžadují a ani nepoužívají prostorový index. Mohou však být stejně jako operátory použity v klauzuli WHERE a kromě toho i v poddotazech.

3.2.6 Prostorové agregační funkce

Agregační funkce v SQL jsou obecně funkce, které se používají k seskupení výsledků SQL dotazu podle nějakého kritéria. V případě Oracle Spatial takové funkce berou v úvahu geometrické objekty a jako výsledek vracejí rovněž geometrický objekt, který je typu SDO_GEOMETRY.

Parametrem mnoha prostorových agregačních funkcí může být objekt typu SDOAGGRTYPE. Nastavením hodnoty atributu `tolerance` tohoto typu lze ovlivnit výsledky některých agregačních funkcí (viz obrázek 3.5).



Obrázek 3.5: Příklad použití tolerance v agregační funkci sjednocení (převzato z [4]).

4 Původní databáze pohybujících se objektů

Už v úvodu bylo zmíněno, že jako základ pro dotazování nad daty pohybujících se objektů měla být použita databáze pohybujících se objektů, tak jak ji navrhl pan Ing. Jaroslav Vališ ve své diplomové práci [2]. Bylo proto nezbytně nutné výše zmíněnou práci a především její výsledek, navrženou databázi pohybujících se objektů, detailně prostudovat a zjistit tak, jaké možnosti nabízí z hlediska dotazování se nad daty pohybujících se objektů. Následujících několik podkapitol je proto věnováno rozboru navrženého řešení panem Ing. Jaroslavem Vališem.

4.1 Model dat

Databáze pohybujících se objektů, tak jak ji pan Ing. Vališ navrhl, byla implementována za použití objektově-relačního modelu dat v databázovém systému Oracle. Data a operace jednotlivých částí struktury databáze jsou zapouzdřeny do uživatelských datových typů. Definované operace odpovídají uloženým procedurám a funkcím databázového systému s jediným rozdílem, a to způsobem přístupu k jednotlivým operacím přes plně kvalifikované jméno názvu uživatelského datového typu, ke kterému náležejí. Jedná se tedy ve všech případech o statické metody tříd navržených datových typů.

Jednotlivé procedury a funkce jsou implementovány v jazyce PL/SQL. Díky tomu je možné využít speciálních rozšíření databázového systému Oracle, zvláště pak podporu práce s prostorovými daty (Oracle Spatial) pro implementaci časoprostorových operací.

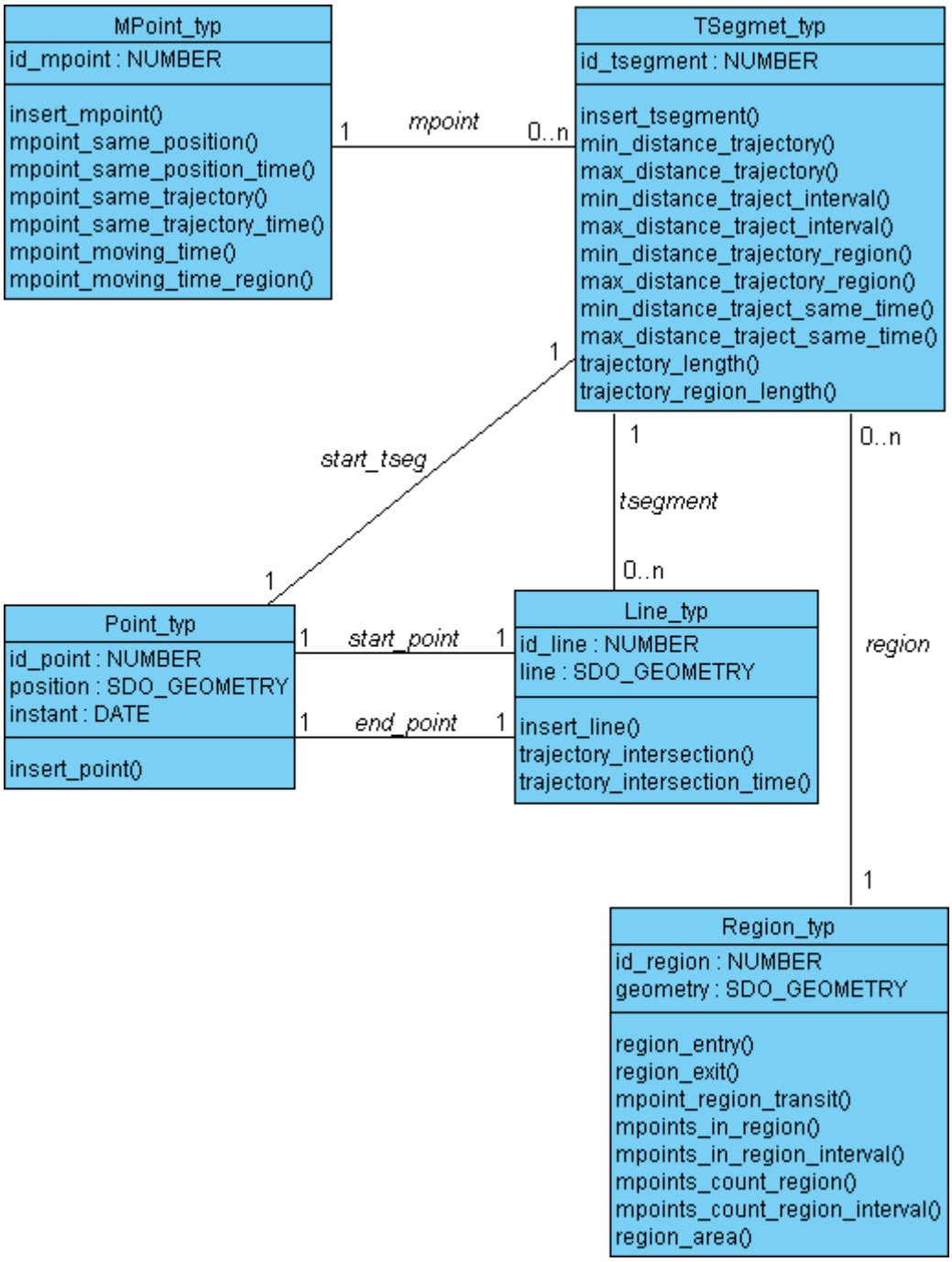
Pro efektivní práci s jednotlivými datovými typy, především pro snadné uložení instancí jednotlivých typů do databáze, jsou navíc u každého datového typu (kromě jediného, viz dále) vytvořeny speciální metody. Ty seskupují skupinu několika SQL příkazů realizujících požadované funkce, které by bylo jinak nutné zadávat později přímo do aplikace, která by takovouto databázi pohybujících se objektů používala.

4.2 Struktura databáze

Hlavní struktura databáze pohybujících se objektů je tedy tvořena pěti uživatelsky definovanými datovými typy. Pomocí nich jsou splněny požadavky na ukládání pohybujících se objektů a také na reprezentaci pohybu těchto objektů v prostoru pomocí trajektorie.

Jednotlivé uživatelské datové typy, jak je patrné na konceptuálním diagramu tříd (obrázek 4.1), jsou mezi sebou silně provázány. Výhodou tohoto modelu je však fakt, že tyto vazby nevznikají až v případě dotazu, ale jsou fyzicky uloženy jako reference na ostatní datové typy v atributech

odkazujícího se datového typu. Tento způsob použití referencí však vede v případě využití tohoto řešení jako databázového rozšíření k nutnosti uložení instance jakéhokoliv definovaného typu do databázových tabulek, a tedy nutnosti správy několika systémových tabulek.



Obrázek 4.1: Konceptuální diagram tříd (převzato z [2]).

Hlavním prvkem struktury je samotný pohybující se objekt. Ten je realizován datovým typem *MPoint_typ*, jehož trajektorie je poskládána ze segmentů (datový typ *TSegment_typ*) odpovídajících té části trajektorie, která popisuje pohyb objektu v určitém omezeném prostoru reprezentovaném datovým typem *Region_typ*. Segmenty trajektorií jsou dále tvořeny úsečkami (typ *Line_typ*), které mají specifikovány počáteční a koncový bod doplněný o časovou informaci. Takovým bodům odpovídá datový typ *Point_typ*.

Z takto vytvořené struktury databáze je později možné snadno získat informace o minulé a současné poloze bodu ve vybraném prostoru.

4.3 Datové typy a operace pro dotazování

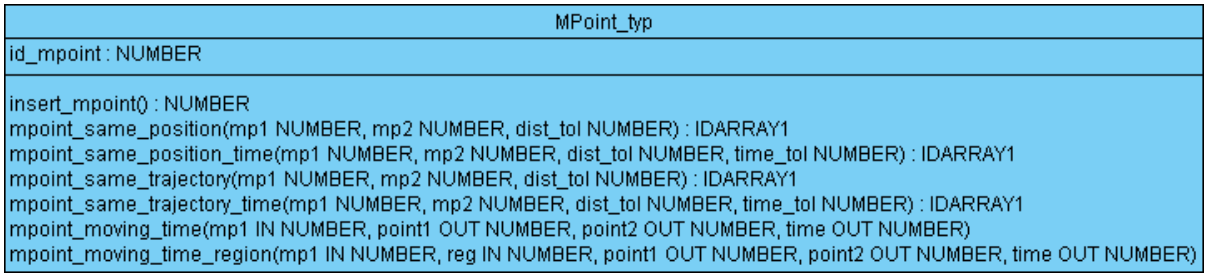
Kromě výše uvedených datových typů, jejichž detaily se budeme zabývat v následujících několika kapitolách, byly v databázi vytvořeny další tři pomocné datové typy. Slouží hlavně jako struktury pro předávání parametrů a výsledků v implementovaných časoprostorových operacích.

K ukládání identifikátorů prostorových entit (bodů a úseček) byly nadefinovány dva datové typy nazvané *IDARRAY1* a *IDARRAY2*. V obou případech se jedná o pole hodnot typu NUMBER.

Třetím pomocným typem je pole objektů typu SDO_GEOMETRY označené jako *GEOMARRAY*. Využívá se zejména pro uchování nově vytvořených prostorových entit, které jsou výsledkem časoprostorových operací.

4.3.1 Datový typ MPoint_typ

Datový typ MPoint_typ představuje pohybující se objekt (složení datového typu viz obrázek 4.2). V datové struktuře obsahující zatím jen jediný atribut uchovávající identifikátor reprezentovaného objektu je stále nechán prostor pro přidání dalších doménově specifických atributů, které mohou nabývat konkrétních vlastností sledovaného objektu (například SPZ u vozidla, číslo autobusové linky apod.).



Obrázek 4.2: Struktura datového typu MPoint_typ (převzato z [2]).

Kromě datových atributů obsahuje typ MPoint_typ i několik následujících statických procedur a funkcí:

- `FUNCTION insert_mpoint RETURN NUMBER`

  Vytvoří a vloží nový pohybující se objekt do databáze a vrátí jeho identifikátor.

- `FUNCTION mpoint_same_position(mp1 NUMBER, mp2 NUMBER, dist_tol NUMBER)RETURN IDARRAY1`

  Vrací množinu bodů z trajektorie objektu zadaného v prvním parametru, ve kterých se oba objekty setkaly. Nastavením parametru *dist_tol*, lze nastavit toleranci pro určení dvou pozic jako shodných. Funkce je kombinací operací *locations* (viz kapitola 2.2.2) a *distance* (viz kapitola 2.2.1).

- `FUNCTION mpoint_same_position_time(mp1 NUMBER, mp2 NUMBER, dist_tol NUMBER, time_tol NUMBER) RETURN IDARRAY1`

  Rozšířením předchozí funkce je možnost určení všech bodů trajektorie, ve kterých se zadané dva objekty nacházely ve stejný čas. V parametru *time_tol* je navíc možné zadat toleranci pro shodné časy v sekundách. Funkce implementuje operace *atperiods* (viz kapitola 2.2.2), *locations* (viz kapitola 2.2.2) a *distance* (viz kapitola 2.2.1).

- `FUNCTION mpoint_same_trajectory(mp1 NUMBER, mp2 NUMBER, dist_tol NUMBER) RETURN IDARRAY1`

  Funkce vrací pole identifikátorů těch úseček trajektorie objektu zadaného v druhém parametru *mp2*, jejichž počáteční a koncový bod se nachází s určitou tolerancí (parametr *dist_tol*) na stejném místě jako počáteční a koncový bod některé z úseček trajektorie objektu zadaného v prvním parametru *mp1*. Funkce je realizována pomocí operací *locations* (viz kapitola 2.2.2), *trajectory* (viz kapitola 2.2.2) a *distance* (viz kapitola 2.2.1).

- `FUNCTION mpoint_same_trajectory_time(mp1 NUMBER, mp2 NUMBER, dist_tol NUMBER, time_tol NUMBER) RETURN IDARRAY1`

  Funkce rozšiřuje předchozí metodu o možnost získání té množiny úseček trajektorie objektu zadaného v parametru *mp2*, která jednak odpovídá společné trajektorii obou zadaných objektů, ale také pro kterou platí, že se objekty v jednotlivých koncových a počátečních bodech úseček vyskytovaly ve stejném čase (s tolerancí zadanou parametrem *time_tol*). Funkce je založena na operacích *atperiods* (viz kapitola 2.2.2), *locations* (viz kapitola 2.2.2), *trajectory* (viz kapitola 2.2.2) a *distance* (viz kapitola 2.2.1).

- `PROCEDURE mpoint_moving_time(mp1 IN NUMBER, point1 OUT NUMBER, point2 OUT NUMBER, time OUT NUMBER)`

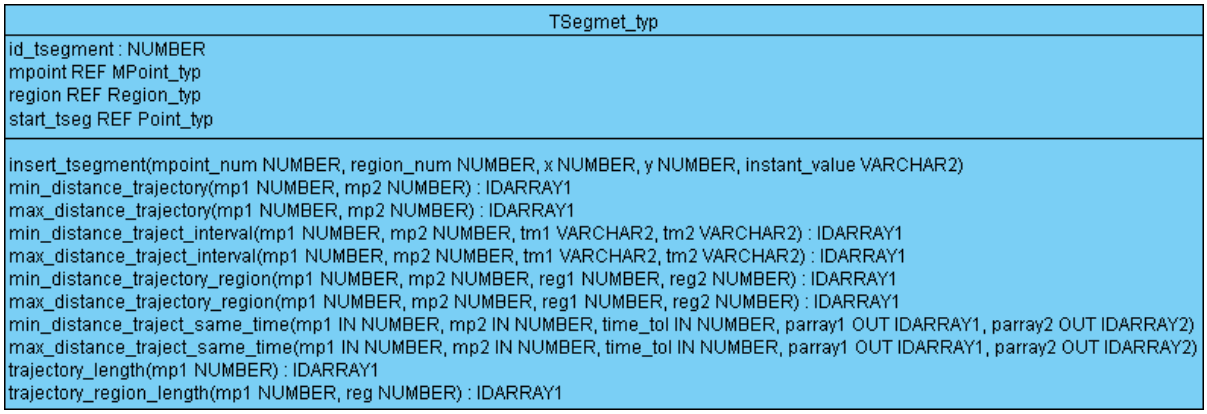
  Procedura vrací celkovou dobu pohybu objektu (výstupní parametr *time*) včetně počátečního (výstupní parametr *point1*) a koncového (výstupní parametr *point2*) bodu jeho trajektorie. Procedura odpovídá operaci *deftime* (viz kapitola 2.2.2).

- `PROCEDURE mpoint_moving_time_region(mp1 IN NUMBER, reg IN NUMBER, point1 OUT NUMBER, point2 OUT NUMBER, time OUT NUMBER)`

  Procedura založená na předchozí, avšak s omezením prostoru pohybu zadaného objektu v parametru *mp1* na region specifikovaný parametrem *reg*. Procedura kombinuje operaci *at* (viz kapitola 2.2.2) s operací *deftime* (viz kapitola 2.2.2).

4.3.2 Datový typ TSegment_typ

Části trajektorie pohybujícího se objektu, které náleží do určitého snímaného prostoru, jsou reprezentovány datovým typem TSegment_typ. Pokaždé, když objekt vstoupí do prostoru, je vytvořen nový segment, u kterého je zároveň uložen vstupní bod do regionu (reprezentovaný datovým typem Point_typ, viz dále). Každý segment tak odpovídá právě jednomu prostoru a jednomu pohybujícímu se objektu. Celá trajektorie pohybu objektu je tedy uspořádanou kolekcí jednotlivých segmentů.



Obrázek 4.3: Struktura datového typu TSegment_typ (převzato z [2]).

Struktura datového typu (viz obrázek 4.3) je tvořena identifikátorem segmentu, dále atributy s odkazy na datové typy reprezentujícími pohybující se objekt a region, ke kterému segment náleží, a vstupní bod segmentu do odpovídajícího regionu. I v tomto případě je datový typ doplněn o následující procedury a funkce:

- `PROCEDURE insert_tsegment(mpoint_num NUMBER, region_num NUMBER, x NUMBER, y NUMBER, instant_value VARCHAR2)`

  Vytvoří a uloží nový segment trajektorie spolu s jeho počátečním bodem datového typu Point_typ (viz dále) odkazovaným atributem *start_tseg*.

- `FUNCTION min_distance_trajectory(mp1 NUMBER, mp2 NUMBER) RETURN IDARRAY1`

  Nalezne minimální vzdálenost trajektorie pohybujícího se objektu *mp2* od trajektorie pohybujícího se objektu *mp1*. Tuto vzdálenost potom vypíše na výstup databáze

DBMS_OUTPUT. Funkce kombinuje operace *locations* (viz kapitola 2.2.2), *trajectory* (viz kapitola 2.2.2) a *distance* (viz kapitola 2.2.1).

- `FUNCTION max_distance_trajectory(mp1 NUMBER, mp2 NUMBER) RETURN IDARRAY1`

  Stejná jako předchozí s tím rozdílem, že hledá maximální vzdálenost trajektorií pohybujících se objektů.

- `FUNCTION min_distance_traject_interval(mp1 NUMBER, mp2 NUMBER, tm1 VARCHAR2, tm2 VARCHAR2) RETURN IDARRAY1`

  Omezuje funkci pro nalezení minimální vzdálenosti trajektorií dvou pohybujících se objektů *mp1* a *mp2* do zadaného časového intervalu. Tento interval je ohraničen počátečním a koncovým časem *tm1* a *tm2*. Implementuje operace *atperiods* (viz kapitola 2.2.2), *locations* (viz kapitola 2.2.2), *trajectory* (viz kapitola 2.2.2) a *distance* (viz kapitola 2.2.1).

- `FUNCTION max_distance_traject_interval(mp1 NUMBER, mp2 NUMBER, tm1 VARCHAR2, tm2 VARCHAR2) RETURN IDARRAY1`

  Stejná jako předchozí funkce, ale na rozdíl od ní hledá maximální vzdálenost v časovém intervalu mezi trajektoriemi dvou pohybujících se objektů.

- `FUNCTION min_distance_trajectory_region(mp1 NUMBER, mp2 NUMBER, reg1 NUMBER, reg2 NUMBER) RETURN IDARRAY1`

  Slouží k nalezení minimální vzdálenosti mezi trajektoriemi pohybujících se objektu *mp1* a *mp2*, které se nacházejí v zadaných regionech *reg1* a *reg2*. Kombinuje operace *at* (viz kapitola 2.2.2), *locations* (viz kapitola 2.2.2), *trajectory* (viz kapitola 2.2.2) a *distance* (viz kapitola 2.2.1).

- `FUNCTION max_distance_trajectory_region(mp1 NUMBER, mp2 NUMBER, reg1 NUMBER, reg2 NUMBER) RETURN IDARRAY1`

  Opět stejná funkce jako předcházející, pouze s tím rozdílem, že hledá maximální vzdálenost trajektorií zadaných pohybujících se objektů *mp1* a *mp2*.

- `PROCEDURE min_distance_traject_same_time(mp1 IN NUMBER, mp2 IN NUMBER, time_tol IN NUMBER, parray1 OUT IDARRAY1, parray2 OUT IDARRAY2)`

  Oproti předchozím funkcím tato procedura slouží k nalezení bodů, ve kterých si ve stejném čase (s tolerancí zadanou parametrem *time_tol*) byly trajektorie pohybujících se objektů nejblíže. Identifikátory takových bodů z obou trajektorií, jsou potom vráceny ve výstupních polích *parray1* a *parray2*. Kromě toho je také hodnota vzdálenosti vypsána na výstup databáze DBMS_OUTPUT. Procedura implementuje operace *atperiods* (viz kapitola 2.2.2), *locations* (viz kapitola 2.2.2), *trajectory* (viz kapitola 2.2.2) a *distance* (viz kapitola 2.2.1).

- `PROCEDURE max_distance_traject_same_time(mp1 IN NUMBER, mp2 IN NUMBER, time_tol IN NUMBER, parray1 OUT IDARRAY1, parray2 OUT IDARRAY2)`

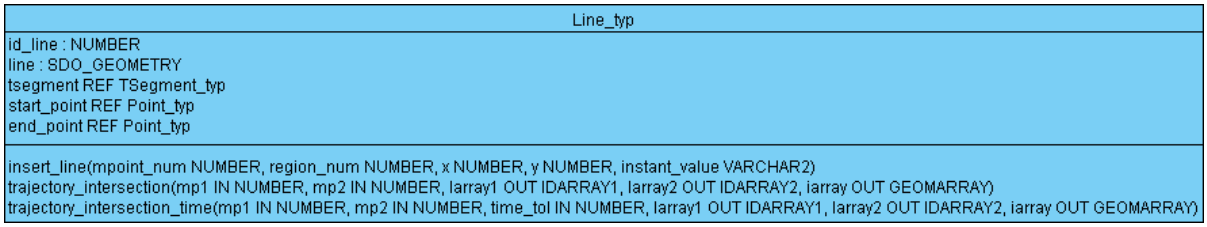
  Funkce odpovídá předchozí, ale opět hledá místo minimální maximální vzdálenost trajektorií pohybujících se objektů ve stejném čase s určitou tolerancí.
- `FUNCTION trajectory_length(mp1 NUMBER) RETURN IDARRAY1`

  Vypočítá celkovou délku trajektorie pohybujícího se objektu zadaného parametrem *mp1*.
- `FUNCTION trajectory_region_length(mp1 NUMBER, reg NUMBER) RETURN IDARRAY1`

  Vypočte délku segmentu trajektorie pohybujícího se objektu *mp1* v zadaném regionu *reg*.

4.3.3 Datový typ Line_typ

Jednotlivé segmenty trajektorií jsou tvořeny množinou úseček. Každá úsečka je specifikována počátečním a koncovým bodem.



Obrázek 4.4: Struktura datového typu Line_typ (převzato z [2]).

Atributy datového typu jsou tedy jednak identifikátor úsečky a odkazy na dva body typu Point_typ, dále odkaz na segment, ke kterému úsečka náleží a navíc také atribut typu SDO_GEOMETRY, který uchovává prostorové informace o úsečce pro efektivní provedení prostorových operací. Struktura tohoto typu (viz obrázek 4.4) je také obohacena o několik níže uvedených procedur:

- `PROCEDURE insert_line(mpoint_num NUMBER, region_num NUMBER, x NUMBER, y NUMBER, instant_value VARCHAR2)`

  Vytvoří a uloží novou úsečku do databáze. Jako počáteční bod uloží koncový bod předchozí úsečky segmentu, nebo v případě, že se jedná o první úsečku segmentu, bod odkazovaný atributem *start_seg* datového typu TSegment_typ. Koncový bod úsečky je pak vytvořen ze zadaných vstupních parametrů.

- `PROCEDURE trajectory_intersection(mp1 IN NUMBER, mp2 IN NUMBER, larray1 OUT IDARRAY1, larray2 OUT IDARRAY2, iarray OUT GEOMARRAY)`

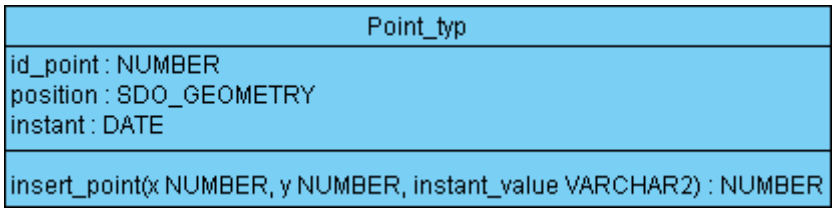
  Nalezne průsečíky trajektorií dvou pohybujících se objektů a vytvoří pro ně nové objekty typu SDO_GEOMETRY s jejich souřadnicemi. Tyto objekty potom vrací ve výstupním poli *iarray*. Kromě toho také vrací pole identifikátorů křížících se úseček jednoho i druhého pohybujícího objektu (výstupní parametry *larray1* a *larray2)*. Procedura implementuje operace *trajectory* (viz kapitola 2.2.2) a *crossings* (viz kapitola 2.2.1).

- `PROCEDURE trajectory_intersection_time(mp1 IN NUMBER, mp2 IN NUMBER, time_tol IN NUMBER, larray1 OUT IDARRAY1, larray2 OUT IDARRAY2, iarray OUT GEOMARRAY)`

  Přidává k předchozí proceduře možnost nalezení průsečíků dvou trajektorií v stejném čase se zadanou tolerancí (vstupní parametr *time_tol*). Výstup procedury je stejný jako v předchozím přípaně. Kombinuje operace *atperiods* (viz kapitola 2.2.2), *trajectory* (viz kapitola 2.2.2) a *crossings* (viz kapitola 2.2.1).

4.3.4 Datový typ Point_typ

Dalším z datových typů, který byl už několikrát zmiňován, je typ Point_typ. Jeho hlavím účelem je zachycení bodu v prostoru a čase. Informace o poloze v prostoru je uchována v atributu, který je typu SDO_GEOMETRY, přičemž časová informace je uložena v atributu typu DATE.



Obrázek 4.5: Struktura datového typu Point_typ (převzato z [2]).

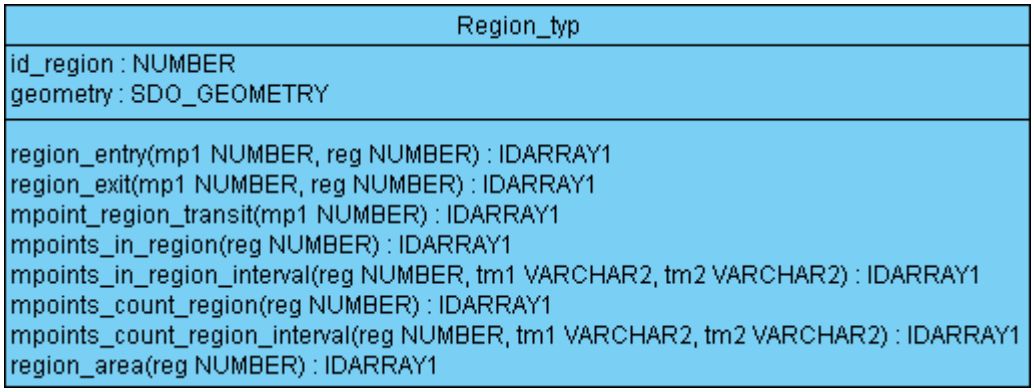
Součástí struktury (viz obrázek 4.5) tohoto datového typu je kromě výše zmíněných atributů, ještě jeden atribut identifikátoru bodu a jedna funkce:

- `FUNCTION insert_point(x NUMBER, y NUMBER, instant_value VARCHAR2)RETURN NUMBER`

  Vrací identifikátor nově uloženého bodu do databáze.

4.3.5 Datový typ Region_typ

Posledním z definovaných uživatelských datových typů je typ Region_typ, který slouží k uchování informace o snímaném prostoru. Jako jediný neobsahuje metodu vytvoření nové instance tohoto typu, protože jeho struktura (viz obrázek 4.6) není nijak provázaná s ostatními datovými typy, a tak je možné vytvářet instance tohoto typu obyčejným SQL příkazem.



Obrázek 4.6: Struktura datového typu Region_typ (převzato z [2]).

Pro práci s tímto datovým typem jsou součástí jeho definice následující funkce:

- `FUNCTION region_entry(mp1 NUMBER, reg NUMBER) RETURN IDARRAY1`
  Výstupem funkce je jednak identifikátor vstupního bodu pohybujícího se objektu (parametr *mp1*) do regionu zadaného v parametru *reg* , a také výpis souřadnic bodu a k němu přiřazenému času na výstup databáze DBMS_OUTPUT. Funkce odpovídá operaci *initial* (viz kapitola 2.2.2).
- `FUNCTION region_exit(mp1 NUMBER, reg NUMBER) RETURN IDARRAY1`
  Funkce, která vrací výstupní bod objektu ze snímaného prostoru podobně jako v předchozím případě. Odpovídá operaci *final* (viz kapitola 2.2.2).
- `FUNCTION mpoint_region_transit(mp1 NUMBER) RETURN IDARRAY1`
  Ve výstupním poli identifikátorů vrací všechny regiony, kterými zadaný pohybující se objekt prošel. Funkce implementuje operaci *passes* (viz kapitola 2.2.2).
- `FUNCTION mpoints_in_region(reg NUMBER) RETURN IDARRAY1`
  Ve výstupním poli identifikátorů vrací všechny pohybující se objekty, které se vyskytovaly v zadaném regionu vstupním parametrem *reg*. Kombinuje operace *locations* (viz kapitola 2.2.2) a operace *intersection* (viz kapitola 2.2.1).
- `FUNCTION mpoints_in_region_interval(reg NUMBER, tm1 VARCHAR2, tm2 VARCHAR2) RETURN IDARRAY1`
  Ve výstupním poli identifikátorů vrací pohybující se objekty, které se v zadaném regionu (vstupní parametr *reg*), vyskytovaly v časovém intervalu ohraničeném parametry *tm1* a *tm2*. Funkce implementuje operace *atperiods* (viz kapitola 2.2.2), *locations* (viz kapitola 2.2.2) a *intersection* (viz kapitola 2.2.1).

- `FUNCTION mpoints_count_region(reg NUMBER) RETURN IDARRAY1`
  Vrací počet pohybujících se objektů, které se vyskytovaly v zadaném regionu (vstupní parametr *reg*). Kombinuje operaci *at* (viz kapitola 2.2.2) s operací *count* (viz kapitola 2.2.1).
- `FUNCTION mpoints_count_region_interval(reg NUMBER, tm1 VARCHAR2, tm2 VARCHAR2) RETURN IDARRAY1`
  Vrací počet pohybujících se objektů, které se vyskytovaly v zadaném regionu (vstupní parametr *reg*) během časového intervalu ohraničeného vstupními parametry *tm1* a *tm2*. Funkce odpovídá kombinaci operací *atperiods* (viz kapitola 2.2.2), *passes* (viz kapitola 2.2.2) a *count* (viz kapitola 2.2.1).
- `FUNCTION region_area(reg NUMBER) RETURN IDARRAY1`
  Vypočte velikost plochy regionu.

4.4 Indexace časoprostorových dat

Pro indexování časoprostorových dat se nabízí několik možností. V důsledku chybějící podpory takových dat v databázovém systému Oracle je ovšem nutné použít indexy buď na data prostorová, nebo na data časová. V diplomové práci pana Ing. Jiřího Vetešníka [3] bylo experimentálně potvrzeno, že optimalizační nástroj pro vykonávání dotazů systému Oracle nelze donutit k užívání obou indexů, tedy časového i prostorového, zároveň. Z hlediska výkonnostního byl v tomto případě zvolen prostorový index, čímž lze dosáhnout vyššího výkonu častých prostorových operací. Pro využití tohoto prostorového indexu je však nutné mít data uložena v databázových tabulkách, nad kterými je vytvořen. To je v tomto případě umožněno způsobem uložení dat pohybujících se objektů a to do předem definovaných databázových tabulek.

5 Navržené časoprostorové rozšíření – Spatio-Temporal Database

Podrobným prostudováním původní databáze pohybujících se objektů, detailně popsané v předchozí kapitole, se ukázalo, že se nejedná o obecnou podporu uložení časoprostorových dat a jejich následné dotazování. Jsou zde k dispozici některé základní operace dotazování, nicméně se jedná o specifické využití formálně zavedeného systému časoprostorových datových typů a operací. Z důvodu pokračování této práce v doktorském studiu bylo nutné vytvořit zcela nový základ časoprostorového rozšíření, které by následovalo formální zavedení této problematiky v kapitole 2 a sloužilo tak jako obecný nástroj pro práci s daty pohybujících se objektů, ať už se jedná o pohybující body, regiony, či pouze v čase se měnící hodnoty nějaké veličiny základního datového typu (např. int).

Výsledkem práce na nové podpoře časoprostorového dotazovaní se tak stala relativně rozsáhlá sada objektových typů, kolekcí a balíčků operací implementovaných v jazyce PL/SQL v prostředí databázového systému Oracle 11g. Pro snadnější označení tohoto databázového rozšíření byl zvolen pracovní název STD, vzniklý z počátečních písmen anglického spojení popisujícího tuto problematiku Spatio-Temporal Database.

Z důvodu použití PL/SQL jako implementačního jazyka je nutné upřesnit konvenci v reprezentaci diagramů jednotlivých částí STD. Jsou zde použity syntaktické prvky jazyka UML, které ovšem v některých případech nemusí odpovídat jejich přesné sémantice vzhledem ke specifikaci UML. Jedná se zejména o databázové balíčky, které jsou reprezentovány pomocí grafického prvku podle UML odpovídajícímu třídě objektu, a to z důvodu možné reprezentace proměnných jako jedné ze součástí balíčku.

5.1 Architektura STD

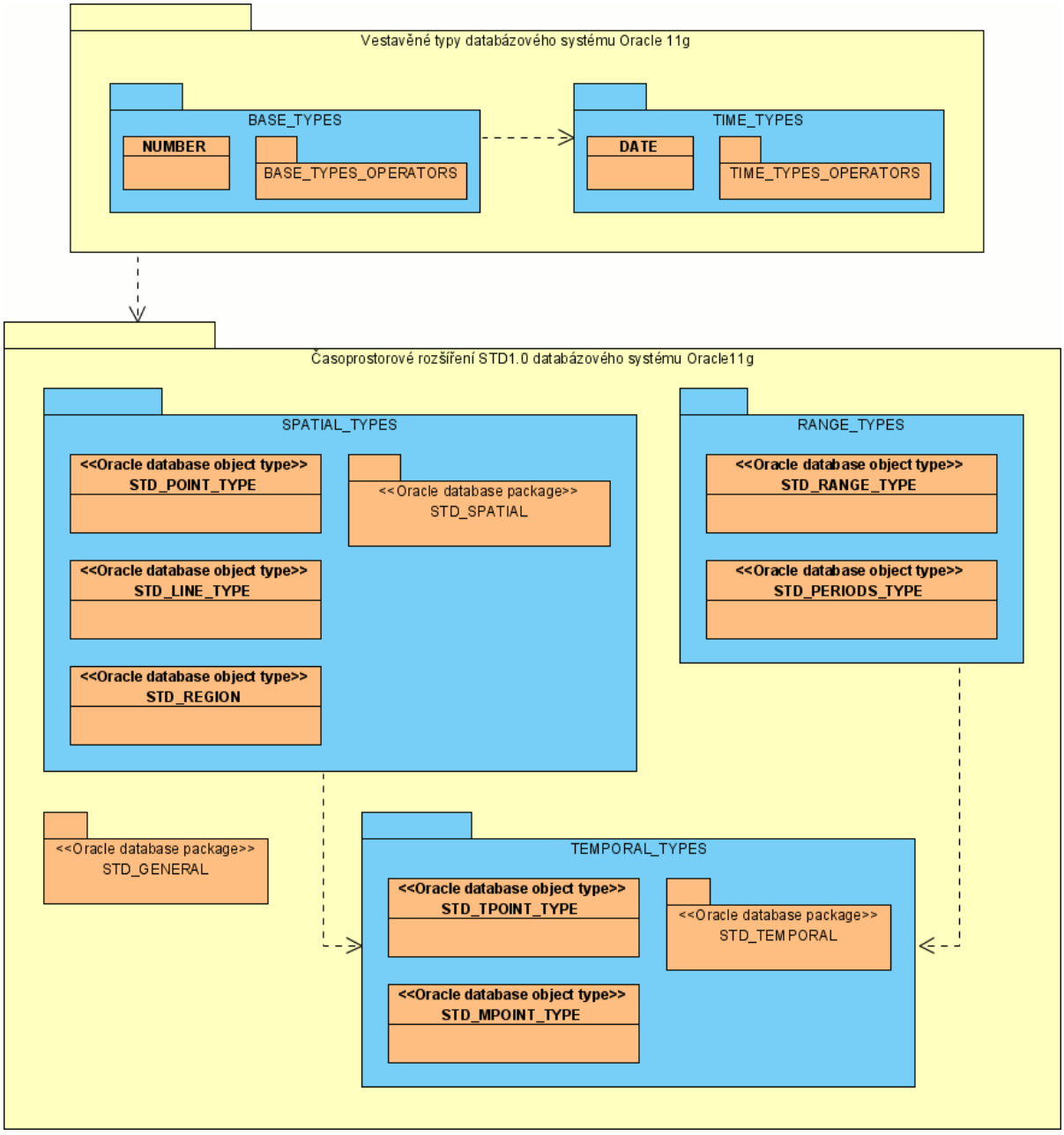
Databázové rozšíření STD je tvořeno, jak už bylo zmíněno, sadou několika objektových typů a balíčků operací, které jsou hierarchicky strukturovány podobně, jak bylo zavedeno v kapitole 2 při formální specifikaci systému časoprostorových datových typů.

Při jeho realizaci bylo využito stejně jako v původně navržené databázi pohybujících se objektů objektově-relační podpory databázového systému Oracle 11g (více o použití objektů viz [8]). Způsob využití této podpory se však v obou řešeních zcela liší.

Kromě základních a časových datových typů, které patří k základní výbavě použitého databázového systému, byly implementovány všechny ostatní skupiny datových typů zavedené v kapitole 2. Jedná se o prostorové datové typy, rozsahové datové typy a datové typy temporální. Ke každému z těchto typů byly dále implementovány balíčky operací (více o použití databázových

balíčků viz [10]), které mají ve skutečnosti funkci sady operátorů použitelných na konkrétní datový typ. Tento způsob implementace přináší hlavní výhodu tohoto řešení a to možnosti velice snadného použití a vzájemného zanořování základních operací přímo při tvorbě jakéhokoliv SQL dotazu a nikoliv pouze v blocích PL/SQL kódu.

Na obrázku 5.1 je zobrazeno rámcové schéma celé architektury STD s naznačenými závislostmi mezi jednotlivými částmi tohoto rozšíření. Použití grafických prvků v tomto schématu stejně jako v případě diagramů zcela neodpovídá specifikaci UML a slouží pouze jako prezentace a logické seskupení jednotlivých částí implementovaného řešení.



Obrázek 5.1: Schéma architektury databázového rozšíření STD.

Kromě částí prezentovaných schématem architektury je navíc z hlediska reálného předávání výsledků zejména prostorových operací databázové rozšíření doplněno o několik následujících kolekcí implementovaných datových typů (v diagramech označeny hranatými závorkami, jako modifikátory typů základních, např. *STD_POINT_TYPE []*). Jejich názvy jsou intuitivní, nicméně pro úplnost je zde jejich výčet uveden (více o použití kolekcí viz [9]):

- STD_POINT_TYPE_ARRAY – kolekce objektů typu STD_POINT_TYPE (viz kapitola 5.2.1).
- STD_LINE_TYPE_ARRAY – kolekce objektů typu STD_LINE_TYPE (viz kapitola 5.2.2).
- STD_REGION_TYPE_ARRAY – kolekce objektů typu STD_REGION_TYPE (viz kapitola 5.2.3).
- STD_TPOINT_TYPE_ARRAY – kolekce objektů typu STD_TPOINT_TYPE (viz kapitola 5.4.1)
- STD_MPOINT_TYPE_ARRAY – kolekce objektů typu STD_MPOINT_TYPE (viz kapitola 5.4.2)

V diagramech tříd datových typů je také někdy uváděn datový typ BOOLEAN, který je však v reálné implementaci nahrazen datovým typem VARCHAR2, protože není podporováno jeho použití v SQL dotazech. Je zde uváděn hlavně z důvodu intuitivnosti diagramů.

5.2 Prostorové datové typy

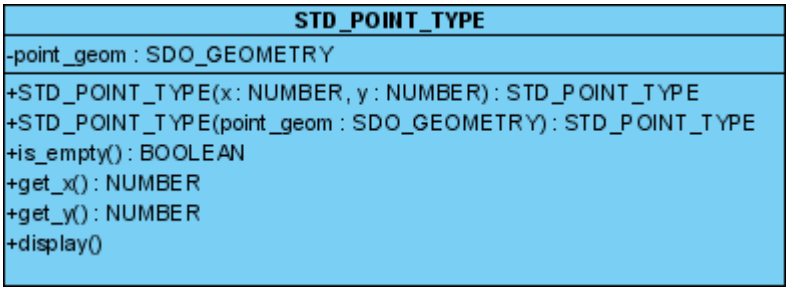
Pro reprezentaci a práci s prostorovými datovými typy bylo využito prostorové rozšíření databázového systému Oracle nazvané Oracle Spatial. Byly implementovány tři základní prostorové objektové datové typy STD_POINT_TYPE, STD_LINE_TYPE a STD_REGION_TYPE.

Ke každému z vytvořených datových typů byly dále implementovány třídní operace sloužící pro práci pouze s objektem konkrétního datového typu (např. získaní délky čáry, či hranice regionu), svým způsobem tedy odpovídají unárním operátorům.

Funkce, které ve skutečnosti odpovídají operátorům nad datovými typy, pak byly seskupeny do databázového balíčku STD_SPATIAL. Ten dává uživateli možnost použití všech základních operací nad prostorovými daty formálně definovaných v kapitole 2.

5.2.1 Datový typ STD_POINT_TYPE

Datový typ STD_POINT_TYPE odpovídá základnímu prostorovému elementu a tím je *bod*. Informace o jeho prostorovém uspořádání si uchovává v atributu *point_geom*, který je typu SDO_GEOMETRY, a to z důvodu snadné realizace prostorových operací využitím databázového rozšíření Oracle Spatial.



Obrázek 5.2: Diagram třídy datového typu STD_POINT_TYPE.

Tento datový typ (diagram třídy datového typu viz obrázek 5.2) dále nabízí několik následujících třídních operací včetně dvou typových konstruktorů:

- `CONSTRUCTOR FUNCTION STD_POINT_TYPE (x NUMBER, y NUMBER) RETURN STD_POINT_TYPE`

  Vytvoří instanci datového typu podle zadaných souřadnic bodu v atributech *x* a *y*.
- `CONSTRUCTOR FUNCTION STD_POINT_TYPE (point_geom SDO_GEOMETRY) RETURN STD_POINT_TYPE`

  Pokusí se vytvořit instanci datového typu podle zadaného objektu typu SDO_GEOMETRY.
- `MEMBER FUNCTION is_empty RETURN BOOLEAN`

  Funkce pro zjištění, zda je objekt datového typu definován. Vrací FALSE, pokud je definován, jinak TRUE.
- `MEMBER FUNCTION get_x RETURN NUMBER`

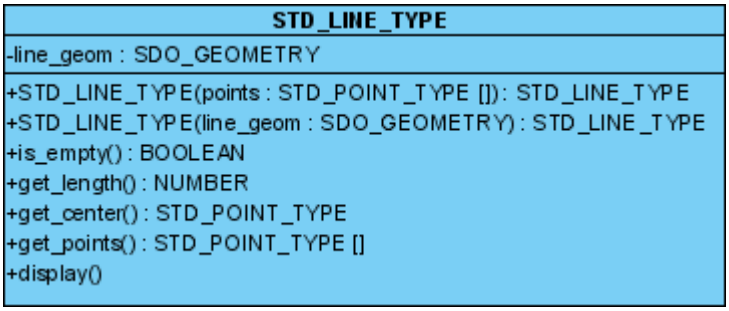
  Získá x-ovou souřadnici bodu.
- `MEMBER FUNCTION get_y RETURN NUMBER`

  Získá y-ovou souřadnici bodu.
- `MEMBER PROCEDURE display`

  Vypíše souřadnice bodu společně s identifikací datového typu na DBMS_OUTPUT.

5.2.2 Datový typ STD_LINE_TYPE

Spojením několika úseček dohromady vzniká prostorový objekt reprezentovaný datovým typem STD_LINE_TYPE (v této práci též někdy označován českým překladem *čára*). Jeho prostorovou informací jsou ve skutečnosti pouze souřadnice bodů označujících začátky a konce úseček, ze kterých je čára složena. Stejně jako v předchozím případě je pro uložení této informace použit jediný atribut datového typu SDO_GEOMETRY nazvaný *line_geom*.



Obrázek 5.3: Diagram třídy datového typu STD_LINE_TYPE.

Pro přímou manipulaci s tímto datovým typem (diagram třídy datového typu viz obrázek 5.3) bylo rovněž implementováno několik třídních operací, také včetně dvou typových konstruktorů:

- `CONSTRUCTOR FUNCTION STD_LINE_TYPE (points STD_POINT_TYPE_ARRAY) RETURN STD_LINE_TYPE`

  Vytvoří instanci datového typu ze zadané kolekce objektů typu STD_POINT_TYPE s body vymezujícími čáru.
- `CONSTRUCTOR FUNCTION STD_LINE_TYPE (line_geom SDO_GEOMETRY) RETURN STD_LINE_TYPE`

  Pokusí se vytvořit instanci datového typu podle zadaného objektu typu SDO_GEOMETRY.
- `MEMBER FUNCTION is_empty RETURN BOOLEAN`

  Funkce pro zjištění, zda je objekt datového typu definován. Vrací FALSE, pokud je definován, jinak TRUE.
- `MEMBER FUNCTION get_length RETURN NUMBER`

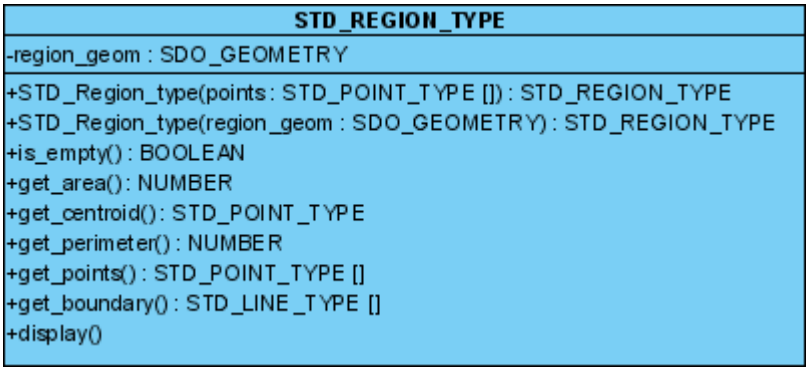
  Získá celkovou délku čáry.
- `MEMBER FUNCTION get_points RETURN STD_POINT_TYPE_ARRAY`

  Získá kolekci objektů typu STD_POINT_TYPE s body vymezujícími čáru.
- `MEMBER PROCEDURE display`

  Vypíše souřadnice bodů vymezujících čáru společně s identifikací datového typu na DBMS_OUTPUT.

5.2.3 Datový typ STD_REGION_TYPE

V hierarchii prostorových datových typů nejvyšším je prostorový objekt *region* (někdy též geometricky *polygon*), kterému odpovídá datový typ STD_REGION_TYPE. Při své definici následuje dva předchozí typy a data své prostorové informace ukládá rovněž pouze do jednoho atributu *region_geom*, který je datového typu SDO_GEOMETRY. Stejně jako v případě čáry se ve skutečnosti jedná o soupis souřadnic bodů, které ho vymezují.



Obrázek 5.4: Diagram třídy datového typu STD_REGION_TYPE.

Doplňkovými třídními operacemi s dvěma typovými konstruktory, které jsou u tohoto datového typu (diagram třídy datového typu viz obrázek 5.4) k dispozici, jsou následující:

- `CONSTRUCTOR FUNCTION STD_REGION_TYPE(`
  `points STD_POINT_TYPE_ARRAY`
  `) RETURN STD_REGION_TYPE`

  Vytvoří instanci datového typu ze zadané kolekce objektů typu STD_POINT_TYPE s body vymezujícími region.
- `CONSTRUCTOR FUNCTION STD_REGION_TYPE (region_geom SDO_GEOMETRY)`
  `RETURN STD_REGION_TYPE`

  Pokusí se vytvořit instanci datového typu podle zadaného objektu typu SDO_GEOMETRY.
- `MEMBER FUNCTION is_empty RETURN BOOLEAN`

  Funkce pro zjištění zda je objekt datového typu definován. Vrací FALSE, pokud je definován, jinak TRUE.
- `MEMBER FUNCTION get_area RETURN NUMBER`

  Získá celkový obsah, který daný region zabírá.
- `MEMBER FUNCTION get_centroid RETURN STD_POINT_TYPE`

  Získá objekt typu STD_POINT_TYPE s bodem, který odpovídá těžišti regionu.
- `MEMBER FUNCTION get_perimeter RETURN NUMBER`

  Získá celkovou délku hranice regionu.
- `MEMBER FUNCTION get_points RETURN STD_POINT_TYPE_ARRAY`

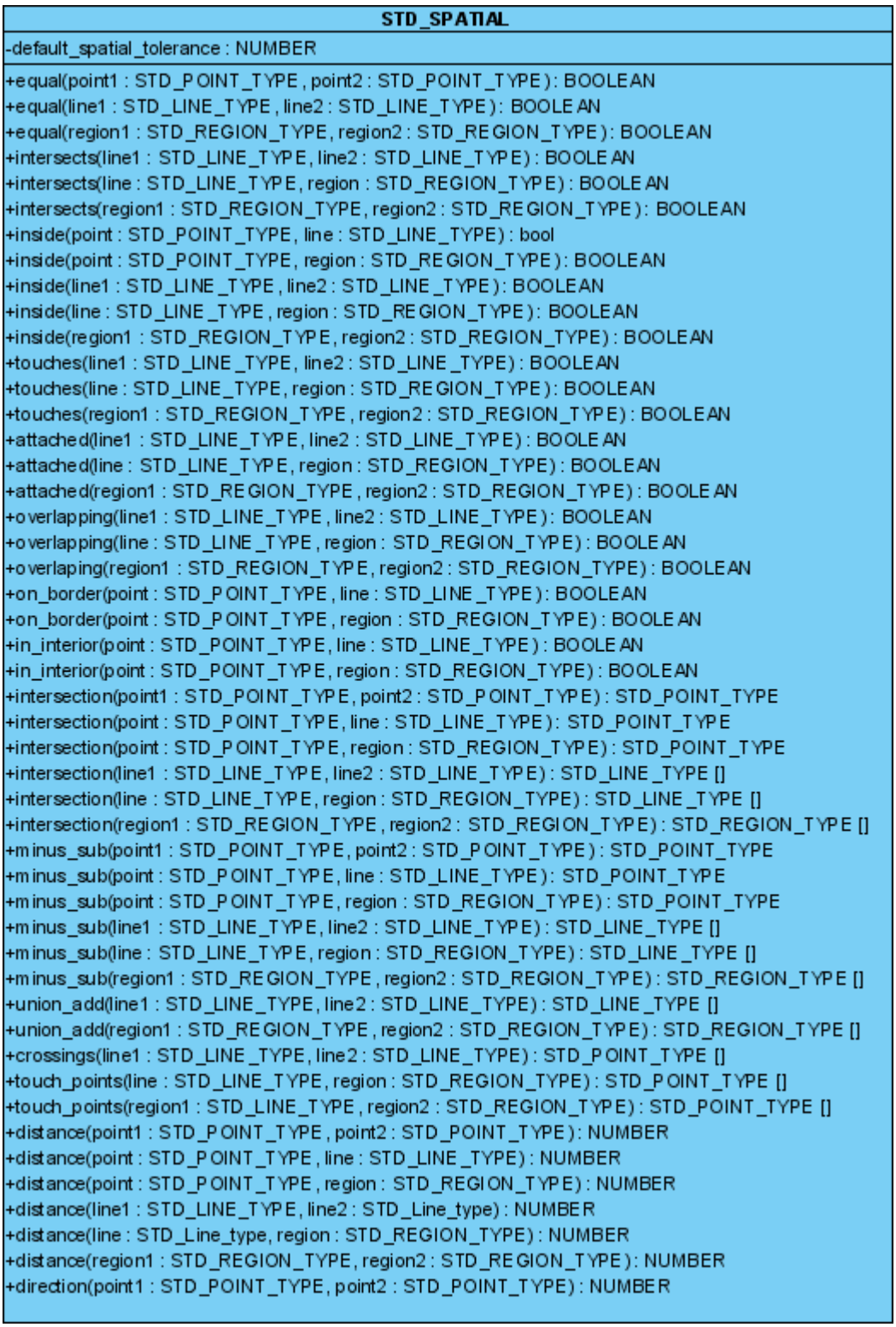
  Získá kolekci objektů typu STD_POINT_TYPE s body vymezujícími region.
- `MEMBER FUNCTION get_boundary RETURN STD_LINE_TYPE_ARRAY`

  Získá pole objektů typu STD_LINE_TYPE s čárami odpovídajícími hranicím regionu.
- `MEMBER PROCEDURE display`

  Vypíše souřadnice bodů vymezujících jednotlivé hranice regionu společně s identifikací datového typu na DBMS_OUTPUT.

5.2.4 Balíček prostorových operací STD_SPATIAL

Rozsáhlou skupinou operací nad prostorovými datovými typy je databázový balíček STD_SPATIAL (diagram databázového balíčku viz obrázek 5.5). Zahrnuje v sobě všechny prostorové operace formálně zavedené v kapitole 2.



Obrázek 5.5: Diagram databázového balíčku STD_SPATIAL.

Jelikož se jedná o operace v prostoru, je potřeba stanovit určitou přesnost těchto operací, tedy stanovit určitý limit, do kdy ještě považujeme dva body v prostoru jako shodné, a od kdy už ne. K tomu je určena *prostorová tolerance*, kterou je možné u každé z implementovaných operací nastavit ručně nebo využít výchozí pro všechny operace balíčku STD_SPATIAL shodnou prostorovou toleranci uchovanou v atributu balíčku STD_SPATIAL *default_spatial_tolerance*.

Díky přetěžování funkcí bylo možné vytvořit jednotně nazvané funkce s různými datovými typy vstupních parametrů. Pro každou z funkcí, jak už bylo naznačeno, byly také navíc implementovány varianty umožňující zadání konkrétní hodnoty prostorové tolerance.

V následujícím textu jsou blíže popsány jednotlivé skupiny operací, tak jak byly implementovány v databázovém rozšíření STD (nejsou zde uvedeny varianty s možností zadání prostorové tolerance):

- ```
  FUNCTION equal (point1 STD_POINT_TYPE, point2 STD_POINT_TYPE)
                  RETURN BOOLEAN
  FUNCTION equal (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)
                  RETURN BOOLEAN
  FUNCTION equal (region1 STD_REGION_TYPE,
                  region2 STD_REGION_TYPE) RETURN BOOLEAN
  ```

  Pro vzájemné prostorové porovnání dvou objektů stejných datových typů slouží skupina funkcí *equal*. Pokud jsou dva objekty shodné, vrací TRUE, jinak FALSE. Shodnost dvou objektů je navíc možné ovlivnit zadáním prostorové tolerance použité ve smyslu uvedeném výše.

- ```
  FUNCTION intersects (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)
                      RETURN BOOLEAN
  FUNCTION intersects (line STD_LINE_TYPE, region STD_REGION_TYPE)
                      RETURN BOOLEAN
  FUNCTION intersects (region1 STD_REGION_TYPE,
                      region2 STD_REGION_TYPE) RETURN BOOLEAN
  ```

  Informaci o tom, zda se dva prostorové objekty protínají či ne podává skupina operací označená názvem *intersects*. Jednotlivé funkce vrací buď TRUE, pokud se objekty protínají, jinak vrací FALSE.

- ```
  FUNCTION inside (point STD_POINT_TYPE, line STD_LINE_TYPE)
                   RETURN BOOLEAN
  FUNCTION inside (point STD_POINT_TYPE, region STD_REGION_TYPE)
                   RETURN BOOLEAN
  FUNCTION inside (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)
                   RETURN BOOLEAN
  FUNCTION inside (line STD_LINE_TYPE, region STD_REGION_TYPE)
                   RETURN BOOLEAN
  ```
  ```

```
FUNCTION inside (region1 STD_REGION_TYPE,
                 region2 STD_REGION_TYPE) RETURN BOOLEAN
```

Další početnou skupinou variant prostorových operací jsou funkce nazvané *inside*. Slouží k zjištění, zda je prostorový objekt zadaný v prvním argumentu operace prostorově obsažen v prostorovém objektu zadaném argumentu druhém.

- ```
  FUNCTION touches (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)
                   RETURN BOOLEAN
  FUNCTION touches (line STD_LINE_TYPE, region STD_REGION_TYPE)
                   RETURN BOOLEAN
  FUNCTION touches (region1 STD_REGION_TYPE,
                   region2 STD_REGION_TYPE) RETURN BOOLEAN
  ```

  K zjištění, zda se dva prostorové objekty dotýkají, tedy průnik jejich hraničních bodů či kontur je neprázdný, jsou určeny operace jednotně označené jako *touches*. Funkce opět vrací buď TRUE, pokud se objekty dotýkají, nebo FALSE, pokud ne.

- ```
  FUNCTION attached (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)
                    RETURN BOOLEAN
  FUNCTION attached (line STD_LINE_TYPE, region STD_REGION_TYPE)
                    RETURN BOOLEAN
  FUNCTION attached (region1 STD_REGION_TYPE,
                    region2 STD_REGION_TYPE) RETURN BOOLEAN
  ```

  Jsou-li dva databázové objekty propojeny, zkoumají funkce nazvané *attached*. Geometricky tato funkce odpovídá zjištění, zda je průnik množiny hraničních bodů nebo kontur prvního objektu a množiny vnitřní části druhého objektu neprázdný. Vrací TRUE, pokud jsou objekty propojeny, jinak vrací FALSE.

- ```
  FUNCTION overlapping (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)
                       RETURN BOOLEAN
  FUNCTION overlapping (line STD_LINE_TYPE,
                       region STD_REGION_TYPE) RETURN BOOLEAN
  FUNCTION overlapping (region1 STD_REGION_TYPE,
                       region2 STD_REGION_TYPE) RETURN BOOLEAN
  ```

  Tato sada operací zjišťuje, zda se dva prostorové objekty překrývají, tedy průnik jejich vnitřků, bez hraničních bodů a kontur, je neprázdná množina. Jako název bylo zvoleno anglické slovo *overlapping*, protože nebylo možné využít původně zavedeného jména základní operace v kapitole 2 *overlaps*, a to z důvodu, že se jedná o jedno z klíčových slov databázového systému Oracle 11g. Funkce rovněž vrací TRUE, pokud se dva objekty překrývají, jinak vrací FALSE.
  ```

- `FUNCTION on_border (point STD_POINT_TYPE, line STD_LINE_TYPE)`
  `RETURN BOOLEAN`
  `FUNCTION on_border (point STD_POINT_TYPE,`
  `region STD_REGION_TYPE) RETURN BOOLEAN`

  Pozicí zadaného prostorového bodu vzhledem k hraničním bodům či konturám prostorové čáry respektive regionu se zabývá skupina funkcí *on_border*. Vrací TRUE, pokud prostorový bod leží na hranici prostorové čáry či regionu.
- `FUNCTION in_interior (point STD_POINT_TYPE, line STD_LINE_TYPE)`
  `RETURN BOOLEAN`
  `FUNCTION in_interior (point STD_POINT_TYPE,`
  `region STD_REGION_TYPE) RETURN BOOLEAN`

  Podobnou množinou operací té předchozí jsou funkce *in_interior* zjišťující naopak, zda je prostorový bod obsažen uvnitř prostorové čáry či regionu. Vrací TRUE, pokud prostorový bod leží v zadané prostorové čáře nebo regionu, jinak vrací FALSE.
- `FUNCTION intersection (point1 STD_POINT_TYPE,`
  `point2 STD_POINT_TYPE) RETURN STD_POINT_TYPE`
  `FUNCTION intersection (point STD_POINT_TYPE, line STD_LINE_TYPE)`
  `RETURN STD_POINT_TYPE`
  `FUNCTION intersection (point STD_POINT_TYPE,`
  `region STD_REGION_TYPE) RETURN STD_POINT_TYPE`
  `FUNCTION intersection (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)`
  `RETURN STD_LINE_TYPE_ARRAY`
  `FUNCTION intersection (line STD_LINE_TYPE,`
  `region STD_REGION_TYPE) RETURN STD_LINE_TYPE_ARRAY`
  `FUNCTION intersection (region1 STD_REGION_TYPE,`
  `region2 STD_REGION_TYPE) RETURN STD_REGION_TYPE_ARRAY`

  Pro získání prostorového objektu nebo množiny prostorových objektů odpovídajících průniku dvou prostorových objektů (možné kombinace jsou uvedeny výše ve funkčních prototypech) slouží skupina funkcí *intersection*. Výsledek operace je vždy datového typu toho z prostorových objektů, který je hierarchicky nižší podle uvedeného uspořádání *bod* < *čára* < *region*, nebo NULL pokud se objekty neprotínají.
- `FUNCTION minus_sub (point1 STD_POINT_TYPE,`
  `point2 STD_POINT_TYPE) RETURN STD_POINT_TYPE`
  `FUNCTION minus_sub (point STD_POINT_TYPE, line STD_LINE_TYPE)`
  `RETURN STD_POINT_TYPE`
  `FUNCTION minus_sub (point STD_POINT_TYPE,`
  `region STD_REGION_TYPE) RETURN STD_POINT_TYPE`

```
FUNCTION minus_sub (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)
          RETURN STD_LINE_TYPE_ARRAY
FUNCTION minus_sub (line STD_LINE_TYPE, region STD_REGION_TYPE)
          RETURN STD_LINE_TYPE_ARRAY
FUNCTION minus_sub (region1 STD_REGION_TYPE,
          region2 STD_REGION_TYPE) RETURN STD_REGION_TYPE_ARRAY
```

Provedení prostorového rozdílu dvou zadaných prostorových objektů je realizováno skupinou operací *minus_sub*. Tento název byl zvolen opět z důvodu existence původního názvu operace *minus* jako klíčového slova databázového systému Oracle 11g. Výsledkem operací je prostorový objekt, který je datového typu toho ze zadaných prostorových objektů, který je hierarchicky nižší podle definovaného uspořádání u předchozí sady operací průniku.

- `FUNCTION union_add (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)`
  `RETURN STD_LINE_TYPE_ARRAY`
  `FUNCTION union_add (region1 STD_REGION_TYPE,`
  `region2 STD_REGION_TYPE) RETURN STD_REGION_TYPE_ARRAY`

  Prostorovému sloučení dvou prostorových objektů, které jsou stejného datového typu je věnována sada funkcí union_add. Speciální název byl zvolen ze stejných důvodů jako v předchozím případě. Operace vrací kolekci prostorových objektů odpovídajících sjednocení těchto objektů.
- `FUNCTION crossings (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)`
  `RETURN STD_POINT_TYPE_ARRAY`

  Kolekci průsečíků, tedy jim odpovídajících prostorových bodů, dvou čar je možné získat operací *crossings*. Vrací buď zmíněnou kolekci prostorových bodů nebo NULL v případě, že se čáry nekříží.
- `FUNCTION touch_points (line STD_LINE_TYPE,`
  `region STD_REGION_TYPE) RETURN STD_POINT_TYPE_ARRAY`
  `FUNCTION touch_points (region1 STD_REGION_TYPE,`
  `region2 STD_REGION_TYPE) RETURN STD_POINT_TYPE_ARRAY`

  Pro získání průsečíků čáry a hraničních kontur regionu nebo hraničních kontur dvou regionů jsou určeny funkce nazvané *touch_points*. Vrací kolekci prostorových bodů odpovídajících nalezeným průsečíkům nebo NULL, pokud se objekty neprotínají.
- `FUNCTION distance (point1 STD_POINT_TYPE, point2 STD_POINT_TYPE)`
  `RETURN NUMBER`
  `FUNCTION distance (point STD_POINT_TYPE, line STD_LINE_TYPE)`
  `RETURN NUMBER`
  `FUNCTION distance (point STD_POINT_TYPE, region STD_REGION_TYPE)`
  `RETURN NUMBER`

```
FUNCTION distance (line1 STD_LINE_TYPE, line2 STD_LINE_TYPE)
                    RETURN NUMBER
FUNCTION distance (line STD_LINE_TYPE, region STD_REGION_TYPE)
                    RETURN NUMBER
FUNCTION distance (region1 STD_REGION_TYPE,
                    region2 STD_REGION_TYPE) RETURN NUMBER
```

K určení vzdálenosti dvou prostorových objektů slouží sada operací nazvaná *distance*. Jednotlivé operace vrací vždy vzdálenost dvou nejbližších bodů z obou zadaných prostorových objektů.

- `FUNCTION direction (point1 STD_POINT_TYPE,`
  `point2 STD_POINT_TYPE) RETURN NUMBER`

  Poslední z implementovaných funkcí je operace *direction*, která určí směrnici přímky procházející dvěma zadanými prostorovými body, vzhledem k ose *x* kartézského souřadného systému, ve kterém jsou zadané prostorové body popsány.

5.3 Rozsahové datové typy

Rozsahové datové typy implementované v databázovém rozšíření STD jsou odvozeny od základních a časových datových typů poskytovaných databázovým systémem Oracle 11g a jsou to STD_RANGE_TYPE a STD_PERIODS_TYPE.

U těchto datových typů byly implementovány pouze třídní funkce ve smyslu unárních operátorů. Balíček funkcí pro práci se dvěma a více instancemi těchto typů zatím není součástí této počáteční verze databázového rozšíření STD.

5.3.1 Datový typ STD_RANGE_TYPE

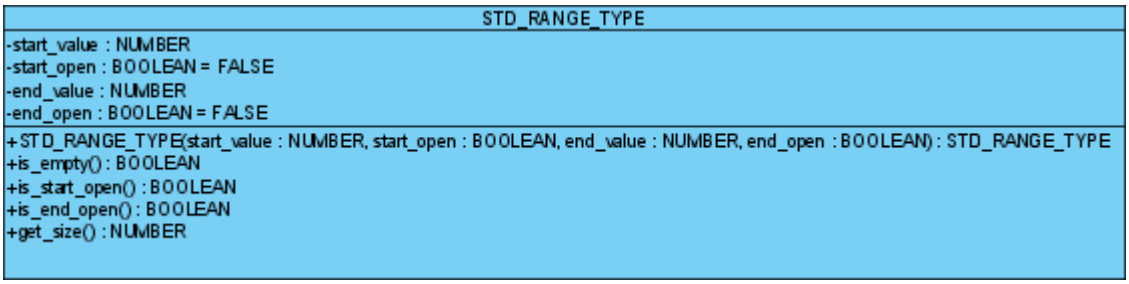
K tomu, aby bylo možné v systému uchovávat různé rozsahy hodnoty, byl implementován datový typ STD_RANGE_TYPE. Odpovídá rozsahu numerických hodnot (základní datový typ databázového systému Oracle 11g NUMBER) ohraničeným počáteční a koncovou hodnotou uloženou v atributech datového typu *start_value* a *end_value*. Pro případ, že by bylo nutné definovat otevřený interval, ať už na jedné či druhé straně rozsahu, jsou přidány speciální atributy *start_open* a *end_open* uchovávající příznak otevřenosti příslušné hranice intervalu.

Pro práci s rozsahem hodnot datového typu STD_RANGE_TYPE (diagram datového typu viz obrázek 5.6) jsou k dispozici tyto třídní funkce a jeden typový konstruktor:

- ```
  CONSTRUCTOR FUNCTION STD_RANGE_TYPE (
                        start_value NUMBER, start_open BOOLEAN,
                        end_value NUMBER, end_open BOOLEAN
                      ) RETURN STD_RANGE_TYPE
  ```
  ```

Ze zadané počáteční a koncové hodnoty s příznaky otevřenosti hranice intervalu vytvoří patřičný rozsah. Pokud jsou počáteční a koncová hodnota přehozeny ve smyslu uspořádání od nejmenší do největší hodnoty, jsou konstruktorem správně seřazeny.

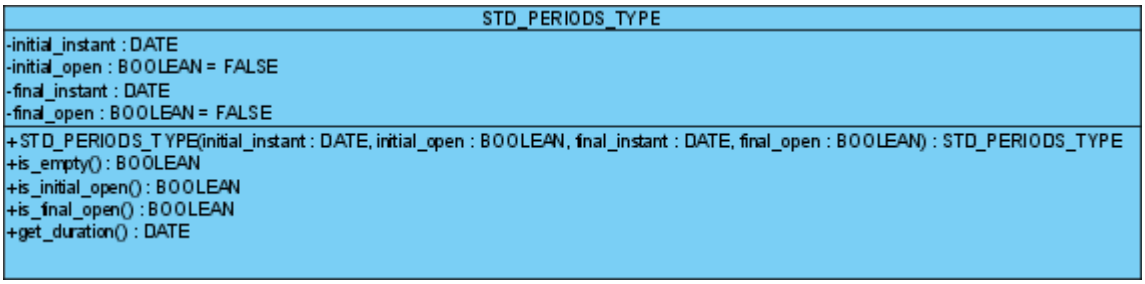
- `MEMBER FUNCTION is_empty RETURN BOOLEAN`
  Funkce pro zjištění, zda je objekt datového typu definován. Vrací FALSE, pokud je definován, jinak TRUE.
- `MEMBER FUNCTION is_start_open RETURN BOOLEAN`
  Funkce pro zjištění, zda je rozsah na své počáteční hranici otevřen. Vrací TRUE, pokud je otevřen, jinak FALSE.
- `MEMBER FUNCTION is_end_open RETURN BOOLEAN`
  Funkce pro zjištění, zda je rozsah na své koncové hranici otevřen. Vrací TRUE, pokud je otevřen, jinak FALSE.
- `MEMBER FUNCTION get_size RETURN NUMBER`
  Získá velikost definovaného rozsahu. Jelikož je konstruktorem zaručeno seřazení intervalu, stačí pouze jednoduché odečtení počáteční hodnoty od koncové.



Obrázek 5.6: Diagram třídy datového typu STD_RANGE_TYPE.

5.3.2 Datový typ STD_PERIODS_TYPE

Téměř identický datový typ s datovým typem STD_RANGE_TYPE je datový typ STD_PERIODS_TYPE (diagram datového typu viz obrázek 5.7). Jedná se pouze o vyčlenění rozsahového datového typu, který definuje rozsah hodnot nad časovou doménou datového typu DATE (rovněž základní datový typ databázového systému Oracle 11g).



Obrázek 5.7: Diagram třídy datového typu STD_PERIODS_TYPE.

Obsahuje také několik operací, které se od původního rozsahového datového typu STD_RANGE_TYPE mírně liší:

- `CONSTRUCTOR FUNCTION STD_PERIODS_TYPE (`
  `initial_instant DATE, initial_open BOOLEAN,`
  `final_instant DATE, final_open BOOLEAN`
  `) RETURN STD_PERIODS_TYPE`

- `MEMBER FUNCTION is_empty RETURN BOOLEAN`
  Funkce pro zjištění, zda je objekt datového typu definován. Vrací FALSE, pokud je definován, jinak TRUE.
- `MEMBER FUNCTION is_initial_open RETURN BOOLEAN`
  Funkce pro zjištění, zda rozsah zahrnuje počáteční časový okamžik. Vrací TRUE, pokud je otevřen, jinak FALSE.
- `MEMBER FUNCTION is_final_open RETURN BOOLEAN`
  Funkce pro zjištění, zda rozsah zahrnuje koncový časový okamžik. Vrací TRUE, pokud je otevřen, jinak FALSE.
- `MEMBER FUNCTION get_duration RETURN NUMBER`
  Získá celkovou dobu trvání definovaného časového intervalu.

5.4 Temporální datové typy

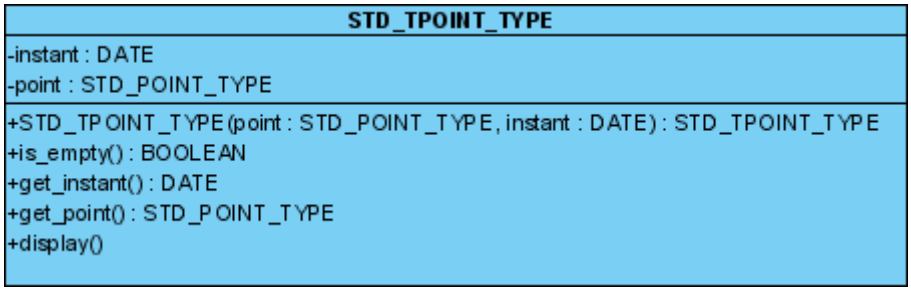
Temporální datové typy, tak jak byly zavedeny v kapitole 2, mapují k hodnotám datových typů základních a prostorových jejich časový vývoj, tedy hodnoty časového datového typu. Od ostatních datových typů, ze kterých jsou odvozeny, je budeme odlišovat přidáním písmene *T* (z anglického *temporal*), označující typy definované jako pár časový okamžik a hodnota příslušného datového typu, a dále *M* (z anlického *moving*) označující typy definované jako konečná množina párů, časový okamžik a hodnota datového typu.

V pojetí této verze databázového rozšíření STD je celkem rozsáhlá skupina temporálních datových typů omezena pouze na dva základní z nich, a to datový typ STD_TPOINT_TYPE a STD_MPOINT_TYPE. V obou případech se jedná o temporální datové typy vztažené k datovému typu STD_POINT_TYPE, tedy bodu definovaném v prostoru. Implementace temporálních typů, jako například STD_TREGION_TYPE, STD_MREGION_TYPE a mnoho dalších, je tak prostorem pro další vývoj STD, na který je však architektura tohoto databázového rozšíření bez problému připravena.

Podobně jako u prostorových datových typů je i zde vytvořena navíc sada základních operací formálně specifikovaných v kapitole 2 označená jako STD_TEMPORAL.

5.4.1 Datový typ STD_TPOINT_TYPE

Jak už bylo uvedeno, velké písmeno *T* v názvu datového typu značí fakt, že se jedná o temporální datový typ definovaný jako pár časový okamžik a hodnota, v tomto případě objekt typu STD_POINT_TYPE, tedy bod v prostoru (v dalším textu je někdy tento pár označován jako *temporální bod*).



Obrázek 5.8: Diagram třídy datového typu STD_TPOINT_TYPE.

Časový okamžik je reprezentován atributem *instant* (datový typ DATE) a bod v prostoru, ke kterému je vztažen, je uchován v atributu *point* (datový typ STD_POINT_TYPE). Součástí datového typu (diagram třídy datového typu viz obrázek 5.8) je i v tomto případě několik následujících třídních operací a jeden typový konstruktor:

- `CONSTRUCTOR FUNCTION STD_TPOINT_TYPE (`
  `instant DATE, point STD_POINT_TYPE`
  `) RETURN STD_TPOINT_TYPE`

  Vytvoří instanci typu STD_TPOINT_TYPE ze zadaného časového okamžiku a objektu typu STD_POINT_TYPE s bodem v prostoru.
- `MEMBER FUNCTION is_empty RETURN BOOLEAN`

  Funkce pro zjištění, zda je objekt datového typu definován. Vrací FALSE, pokud je definován, jinak TRUE.
- `MEMBER FUNCTION get_instant RETURN DATE`

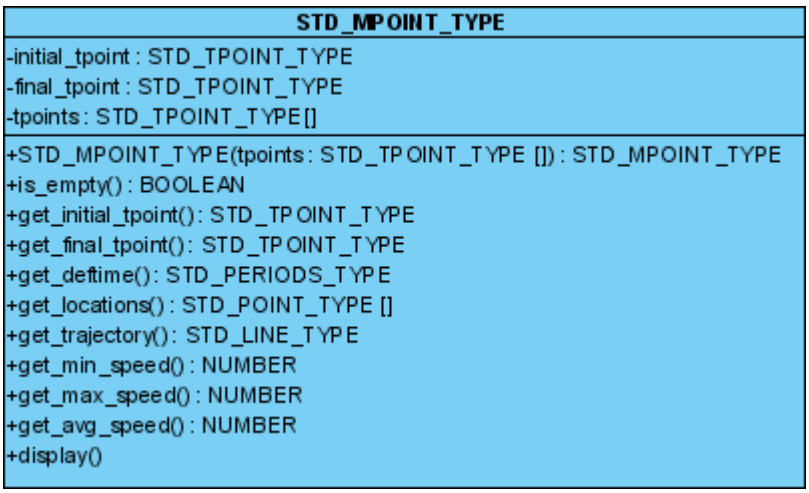
  Získá hodnotu časového okamžiku temporálního bodu.
- `MEMBER FUNCTION get_point RETURN STD_POINT_TYPE`

  Získá objekt typu STD_POINT_TYPE s bodem v prostoru, ke kterému je vztažen daný temporální bod.
- `MEMBER PROCEDURE display`

  Vypíše souřadnice bodu v prostoru spolu s hodnotou časového okamžiku a identifikací datového typu na DBMS_OUTPUT.

5.4.2 Datový typ STD_MPOINT_TYPE

Podle uvedené definice tohoto druhu datového typu (velké písmeno *M* v názvu) se jedná o konečnou množinu párů časový okamžik a hodnota. Jelikož se jedná o temporální datový typ vztažený k bodu v prostoru, jedná se vlastně o množinu objektů předchozího datového typu STD_TPOINT_TYPE. Pozicím bodu v prostoru jsou přiřazeny hodnoty časových okamžiků, což odpovídá diskretizaci spojitého pohybu bodu v prostoru a čase. Tento fakt vede k častému označení instance tohoto datového typu jako *pohybujícího se bodu* (v případě, že pojmem *objekt* značíme pouze hmotný bod v prostoru, je možné použít i spojení *pohybující se objekt*).



Obrázek 5.9: Diagram třídy datového typu STD_MPOINT_TYPE.

Datový typ STD_MPOINT_TYPE (diagram třídy datového typu viz obrázek 5.9) je tedy definovaný pomocí atributu *tpoints* odpovídajícímu podle časového okamžiku seřazené kolekci objektů datového typu STD_TPOINT_TYPE spolu s doplňujícími atributy *initial_tpoint* a *final_tpoint*, které jsou oba datového typu STD_TPOINT_TYPE a představují počáteční a koncový temporální bod. Mezi doplňující operace tohoto datového typu patří:

- `CONSTRUCTOR FUNCTION STD_MPOINT_TYPE (`
  `tpoints STD_TPOINT_TYPE_ARRAY`
  `) RETURN STD_MPOINT_TYPE`

  Vytvoří ze zadané kolekce objektů datového typu STD_TPOINT_TYPE pohybující se bod. Před uložením této kolekce do patřičných atributů datového typu zajistí uspořádání zadané kolekce podle časového okamžiku od nejmenšího k největšímu a odstraní případné duplicity (jeden bod nemůže existovat v jednom časovém okamžiku na dvou pozicích v prostoru).
- `MEMBER FUNCTION is_empty RETURN BOOLEAN`

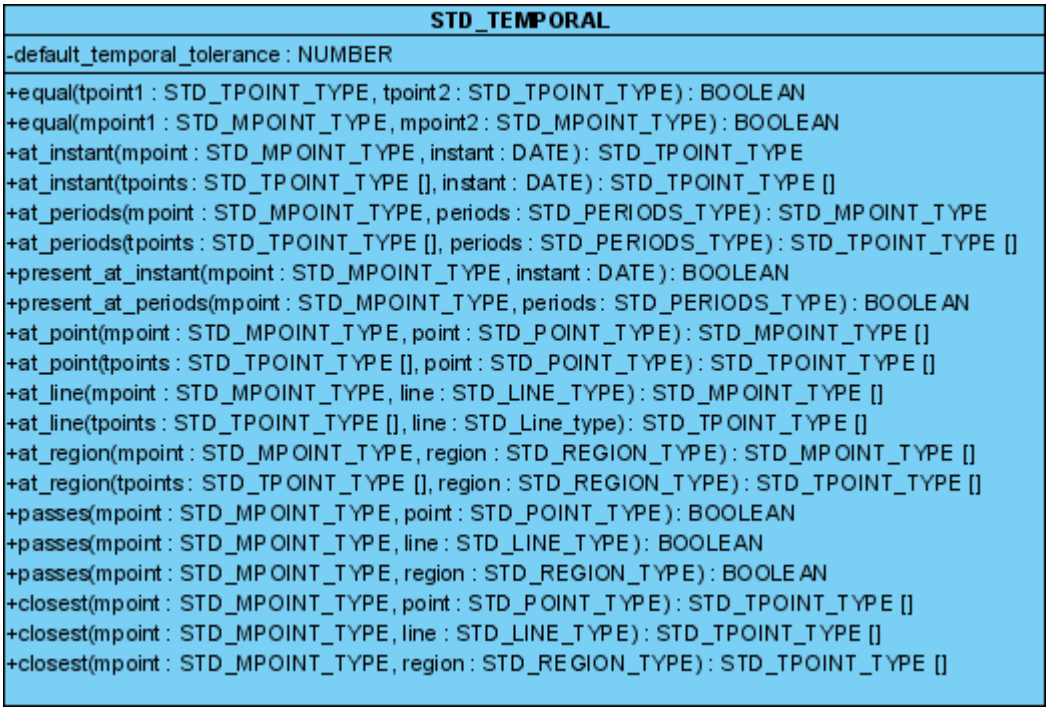
  Funkce pro zjištění, zda je objekt datového typu definován. Vrací FALSE, pokud je definován, jinak TRUE.

- `MEMBER FUNCTION get_initial_tpoint RETURN STD_TPOINT_TYPE`
  Získá objekt typu STD_TPOINT_TYPE s počátečním temporálním bodem pohybujícího se bodu.
- `MEMBER FUNCTION get_final_tpoint RETURN STD_TPOINT_TYPE`
  Získá objekt typu STD_TPOINT_TYPE s koncovým temporálním bodem pohybujícího se bodu.
- `MEMBER FUNCTION get_deftime RETURN STD_PERIODS_TYPE`
  Získá objekt typu STD_PERIODS_TYPE s časovým rozsahem, během kterého je pohybující se bod definován.
- `MEMBER FUNCTION get_locations RETURN STD_POINT_TYPE_ARRAY`
  Získá kolekci objektů typu STD_POINT_TYPE s body v prostoru, ve kterých je pohybující se bod definován.
- `MEMBER FUNCTION get_trajectory RETURN STD_LINE_TYPE`
  Získá objekt typu STD_LINE_TYPE odpovídající trajektorii pohybujícího se bodu.
- `MEMBER FUNCTION get_min_speed RETURN NUMBER`
  Získá minimální rychlost pohybujícího se bodu mezi dvěma následujícími temporálními body.
- `MEMBER FUNCTION get_max_speed RETURN NUMBER`
  Získá maximální rychlost pohybujícího se bodu mezi dvěma následujícími temporálními body.
- `MEMBER FUNCTION get_avg_speed RETURN NUMBER`
  Získá průměrnou rychlost pohybujícího se bodu po dobu jeho existence.
- `MEMBER PROCEDURE display`
  Vypíše souřadnice a časové okamžiky temporálních bodů tvořících pohybující se bod s identifikací datového typu na DBMS_OUTPUT.

5.4.3 Balíček temporálních operací STD_TEMPORAL

Nejrozsáhlejší částí temporálních datových typů je, tak jak tomu bylo i u prostorových typů, databázový balíček operací STD_TEMPORAL (diagram databázového balíčku viz obrázek 5.10). Jsou zde implementovány všechny základní operace definované v kapitole 2, nicméně jak bylo popsáno výše, jsou tyto operace omezeny pouze na zpracování prostorového bodu vztaženého k časové doméně.

V souvislosti s definovanými časoprostorovými operacemi je navíc nutné kromě již zavedené prostorové tolerance, zavést i toleranci časovou. Její specifikaci je možné provést stejným způsobem jako tomu je u prostorové tolerance, tedy buď přímo zadáním její hodnoty jako vstupního parametru každé z operací, nebo použitím výchozí temporální tolerance definované v atributu *default_temporal_tolerance* databázového balíčku STD_TEMPORAL.



Obrázek 5.10: Diagram databázového balíčku STD_TEMPORAL.

Skupiny operací, které je možné využít, jsou blíže popsány v následujícím textu:

- `FUNCTION equal(tpoint1 STD_TPOINT_TYPE, tpoint2 STD_TPOINT_TYPE)`
  `RETURN BOOLEAN`
  `FUNCTION equal(mpoint1 STD_MPOINT_TYPE, mpoint2 STD_MPOINT_TYPE)`
  `RETURN BOOLEAN`

  Pro porovnání dvou temporálních či pohybujících se bodů slouží skupina funkcí nazvaná *equal*. Pokud jsou dva objekty stejného datového typu shodné, funkce vrací TRUE, jinak FALSE.
- `FUNCTION at_instant(mpoint STD_MPOINT_TYPE, instant DATE)`
  `RETURN STD_TPOINT_TYPE`
  `FUNCTION at_instant(tpoints STD_TPOINT_TYPE_ARRAY, instant DATE)`
  `RETURN STD_TPOINT_TYPE_ARRAY`

  K omezení množiny temporálních bodů časovým okamžikem je určena skupina funkcí *at_instant*. V případě pohybujícího se bodu, u kterého je jeho konstruktorem zaručena unikátnost časových okamžiků v jeho průběhu, je vrácen pouze jediný temporální bod, pokud takový v zadaném čase existuje. U nezávislé množiny temporálních bodů, pak může být výsledkem hledaných bodů několik, proto je vráceno pole temporálních bodů. V obou případech, není-li temporální bod se zadaným časovým okamžikem nalezen, je vrácena hodnota *NULL*.
- `FUNCTION at_periods(mpoint STD_MPOINT_TYPE,`
  `periods STD_PERIODS_TYPE) RETURN STD_MPOINT_TYPE`
  `FUNCTION at_periods(tpoints STD_TPOINT_TYPE_ARRAY,`
  `periods STD_PERIODS_TYPE) RETURN STD_TPOINT_TYPE_ARRAY`

Podobnou skupinou operací té předchozí jsou funkce označené *at_periods*, které slouží k omezení pohybujícího se bodu nebo množiny temporálních bodů zadaným časovým rozmezím. Výsledkem této operace je v případě pohybujícího se bodu jeho část, tedy rovněž pohybující se bod. U množiny temporálních bodů se pak jedná o její podmnožinu.

- `FUNCTION present_at_instant(mpoint STD_MPOINT_TYPE, instant DATE) RETURN BOOLEAN`

  K zjištění, zda pohybující se bod existoval v určitém časovém okamžiku, slouží funkce *present_at_instant*. Vrací TRUE pokud ano, jinak je návratovou hodnotou FALSE.

- `FUNCTION present_at_periods(mpoint STD_MPOINT_TYPE, periods STD_PERIODS_TYPE) RETURN BOOLEAN`

  Obdobou předchozí funkce je operace *present_at_periods* zjišťující existenci pohybujícího se bodu v zadaném časovém intervalu. Vrací TRUE pokud pohybující se bod v zadaném rozmezí existoval, nebo FALSE, pokud ne.

- ```
  FUNCTION at_point(mpoint STD_MPOINT_TYPE, point STD_POINT_TYPE)
              RETURN STD_MPOINT_TYPE_ARRAY
  FUNCTION at_point(tpoints STD_TPOINT_TYPE_ARRAY,
              point STD_POINT_TYPE) RETURN STD_TPOINT_TYPE_ARRAY
  FUNCTION at_line(mpoint STD_MPOINT_TYPE, line STD_LINE_TYPE)
              RETURN STD_MPOINT_TYPE_ARRAY
  FUNCTION at_line(tpoints STD_TPOINT_TYPE_ARRAY,
              line STD_LINE_TYPE) RETURN STD_TPOINT_TYPE_ARRAY
  FUNCTION at_region(mpoint STD_MPOINT_TYPE,
              region STD_REGION_TYPE) RETURN STD_MPOINT_TYPE_ARRAY
  FUNCTION at_region(tpoints STD_TPOINT_TYPE_ARRAY,
              region STD_REGION_TYPE) RETURN STD_TPOINT_TYPE_ARRAY
  ```

  Mimo omezení časem, je samozřejmě možné pohybující se body nebo množinu temporálních bodů omezit i některým z prostorových objektů, jako je bod, čára či region. U této skupiny funkcí však nebylo možné použít název funkce *at*, definované v kapitole 2, protože se jedná o jedno z klíčových slov databázového systému Oracle 11g. Jednotlivé varianty této skupiny operací tak bylo nutné oddělit přidáním patřičného sufixu k základnímu názvu funkce (např. „at“ + „_line“).

Výsledkem tohoto druhu operací je u pohybujícího se bodu vždy pole pohybujících se bodů odpovídajících jeho částem, které přísluší danému prostorovému omezení (i u omezení bodem je nutné vracet pohybující se objekt, protože je možné, že se původní pohybující se bod zrovna v tomto místě nepohybuje, ale čas stále pokračuje). V případě množiny temporálních bodů se opět jedná pouze o její podmnožinu odpovídající zadanému prostorovému objektu.
  ```

- `FUNCTION passes(mpoint STD_MPOINT_TYPE, point STD_POINT_TYPE)`
  `RETURN BOOLEAN`
  `FUNCTION passes(mpoint STD_MPOINT_TYPE, line STD_LINE_TYPE)`
  `RETURN BOOLEAN`
  `FUNCTION passes(mpoint STD_MPOINT_TYPE, region STD_REGION_TYPE)`
  `RETURN BOOLEAN`

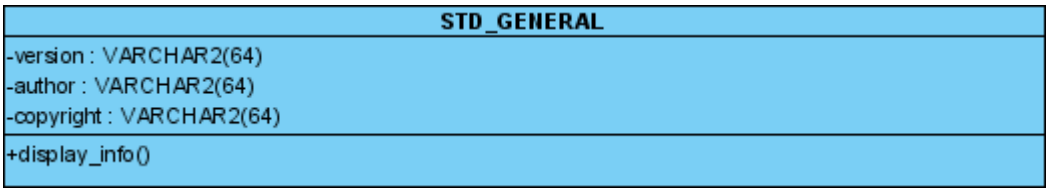
  Zjištěním, zda pohybující se bod procházel během svého „života“ některým ze zadaných prostorových objektů, se zabývá skupina funkcí *passes*. Jednotlivé funkce vrací TRUE, pokud pohybující se bod procházel zadaným prostorovým objektem, jinak FALSE.
- `FUNCTION closest(mpoint STD_MPOINT_TYPE, point STD_POINT_TYPE)`
  `RETURN STD_TPOINT_TYPE_ARRAY`
  `FUNCTION closest(mpoint STD_MPOINT_TYPE, line STD_LINE_TYPE)`
  `RETURN STD_TPOINT_TYPE_ARRAY`
  `FUNCTION closest(mpoint STD_MPOINT_TYPE, region STD_REGION_TYPE)`
  `RETURN STD_TPOINT_TYPE_ARRAY`

  K získání množiny nejbližších míst trajektorie pohybujícího se bodu a zadaného prostorového objektu je určena sada funkcí nazvaná *closest*. Vrací pole temporálních bodu, které leží v nejmenší vzdálenosti k zadanému prostorovému objektu.

5.5 Další součásti

5.5.1 Balíček doplňkových operací STD_GENERAL

STD je možné dále rozšiřovat a přidávat k němu další a další doplňkové části. V této první verzi je této možnosti využito pouze jednoduše a to vytvořením databázového balíčku STD_GENERAL, který slouží jako informační část STD. Uživatel ho tak může využít například k získání verze, autora, licence, či zobrazení formátovaného výstupu obsahující veškeré základní informace o databázovém rozšíření STD (diagram databázového balíčku viz obrázek 5.11).



Obrázek 5.11: Diagram databázového balíčku STD_GENERAL.

5.6 Indexování časoprostorových dat

Jak už bylo naznačeno v kapitole 4 při rozboru původní databáze pohybujících se objektů, tak pro indexování časoprostorových dat se nabízí dvě možnosti. Buď indexování časové části informace, nebo prostorové. Použití obou indexů, jak se ukázalo v diplomové práci pana Ing. Jiřího Vetešníka [3], není možné. Z pohledu častých prostorových operací by se tak zdála vhodnější varianta použití prostorového indexu. Ten s sebou však nese některá omezení, mezi která patří hlavně nutnost specifikace rozměrů prostoru a uložení časoprostorových dat do databázových tabulek, nad kterými je tento index následně vytvořen a upravován. Z hlediska časoprostorových dat, kdy je zřejmé časté vkládání velkého množství dat odpovídajících uložení prostorové informace v každém časovém okamžiku, by však byla aktualizace prostorového indexu neúnosná. Vytvoření tabulek pro uložení dat pohybujících se objektů, tak jak tomu je v původní databázi pohybujících se objektů, by také vedlo k mísení dat různých aplikací využívajících tohoto databázového rozšíření v rámci jednoho databázového systému a tak další výkonnostní limity v použití prostorového indexu.

Implementované databázové rozšíření STD se tak od použití jakéhokoliv databázového indexu, zejména pak prostorového, snaží vyvarovat. Jako jedním z dalších cílů, kterými bude navázáno na tuto diplomovou práci, se však pokusí na základě výsledků diplomové práce Bc. Martiny Křížové [7] na téma „Indexování dat pohybujících se objektů“ vytvořit databázový index speciální pro zpracovávanou doménu časoprostorových dat podobně jako je tomu u databázového rozšíření Oracle Spatial.

Takový časoprostorový index by pak mohl vypadat jako vztažení prostorového indexu, ať už *R-stromu* či *Quad-stromu*, k časovému okamžiku. Jednotlivé prostorové indexy by tak bylo možné navíc indexovat podle svých časových okamžiků, čímž bychom využily maximální možné indexace časoprostorových dat. Samotné prostorové indexy by se tedy vztahovaly pouze na jednotlivý snímek prostorové informace v čase a při vkládání dat nových by tak docházelo k aktualizaci pouze toho prostorového indexu, který přísluší danému časovému okamžiku. Tento postup řešení problému indexace časoprostorových dat je však pouze námětem k další diskuzi na téma časoprostorového indexu a není nikterak ověřen experimenty či reálnou implementací.

6 Ukázková aplikace využívající STD

Za účelem prezentace funkčnosti využití navrženého a implementovaného databázového rozšíření STD byla vytvořena jednoduchá ukázková aplikace. Jedná se o aplikaci z oblasti dohledových systémů zabývající se sledováním pohybujících se objektů ve stanicích metra. Proto byla také aplikace nazvána pracovně *Metro Station Tracker*.

6.1 Popis řešení aplikace

Kvůli snadné přenositelnosti a víceméně žádným dalším nárokům na software osobních počítačů uživatelů, bylo zvoleno řešení ukázkové aplikace jako aplikace webové. Kromě využití databázového serveru Oracle 11g s nainstalovaným databázovým rozšířením STD 1.0, byly technologiemi použitými pro její realizaci jednak Java EE (více o této technologii viz [12]) pro implementaci základní logiky aplikace a komunikace s databázovým serverem a dále klientský JavaScript, pomocí něhož byla jako součást řešení této aplikace implementována sada nástrojů pro vykreslení výsledků a speciální ovládání některých prostorových operací.

Pro vytvoření výsledné podoby aplikace bylo využito možnosti zavedení stylů pomocí CSS a její vzhled a funkčnost byly testovány v internetových prohlížečích Firefox 3.0.10, Internet Explorer 7 a Opera 9.64.

6.1.1 Architektura aplikace

Aplikace Metro Station Tracker, jak už bylo uvedeno je tvořena jako Java EE aplikace s využitím JSP pro vytvoření prezentace jednotlivých stránek a pro komunikaci s databázovým systémem Oracle 11g využívá ovladač JDBC (více o tomto ovladači viz [13]).

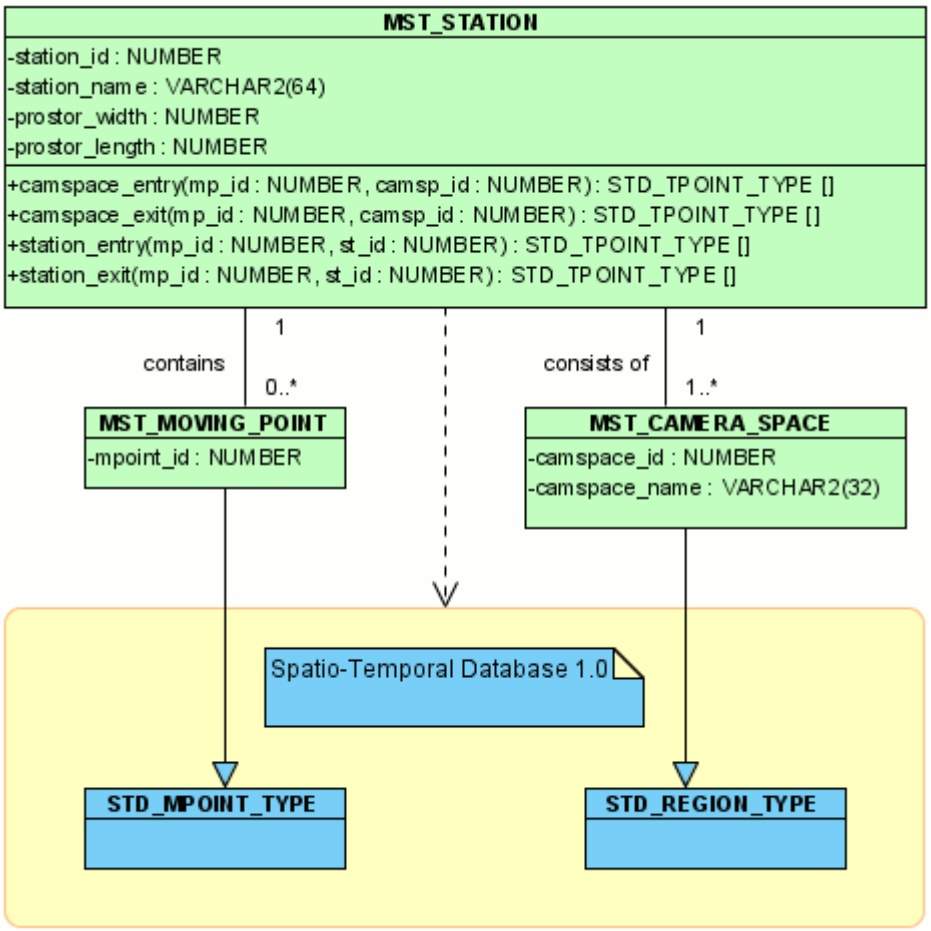
Při definici struktury databáze bylo využito hlavní výhody STD, a to možnosti vytvoření aplikačně specifických datových typů zděděním některého z datových typů definovaných v STD. Tyto nové datové typy tak bylo možné rozšířit o další atributy potřebné k uložení informací o objektech z oblasti dohledových systémů, to vše však při zachování plné podpory časoprostorového dotazování (konceptuální schéma odvozených datových typů viz obrázek 6.1). Na základě těchto nově definovaných datových typů byly následně vytvořeny databázové tabulky, které potom byly mezi sebou vzájemně provázány. Takto vytvořená výsledná architektura se mimo jiné také nápadně podobá architektuře původně navržené databáze pohybujících se objektů panem Ing. Jaroslavem Vališem, čímž se potvrzuje tvrzení, že se jedná spíše o konkrétní aplikaci podpory pro práci s časoprostorovými daty, než o obecnou podporu zpracování časoprostorových dat.

Jak je patrné z konceptuálního diagramu na obrázku 6.1, slouží k uložení informací o jednoduchém dohledovém systému stanic metra tři základní množiny databázových objektů.

Prvním a zároveň nejdůležitějším objektem je objekt popisující samotnou stanici metra, její název a rozměry plochy, kterou zabírá (tento údaj je nutné uchovat kvůli grafickému prvku uživatelského rozhraní pro zobrazení prostorových informací vztahujících se k jednotlivým stanicím metra).

Druhým typem objektů jsou objekty odpovídající regionům pokrytí kamerami dohledového systému přiřazeným k jednotlivým stanicím metra (každá ze stanic musí obsahovat alespoň jedno kamerové pokrytí).

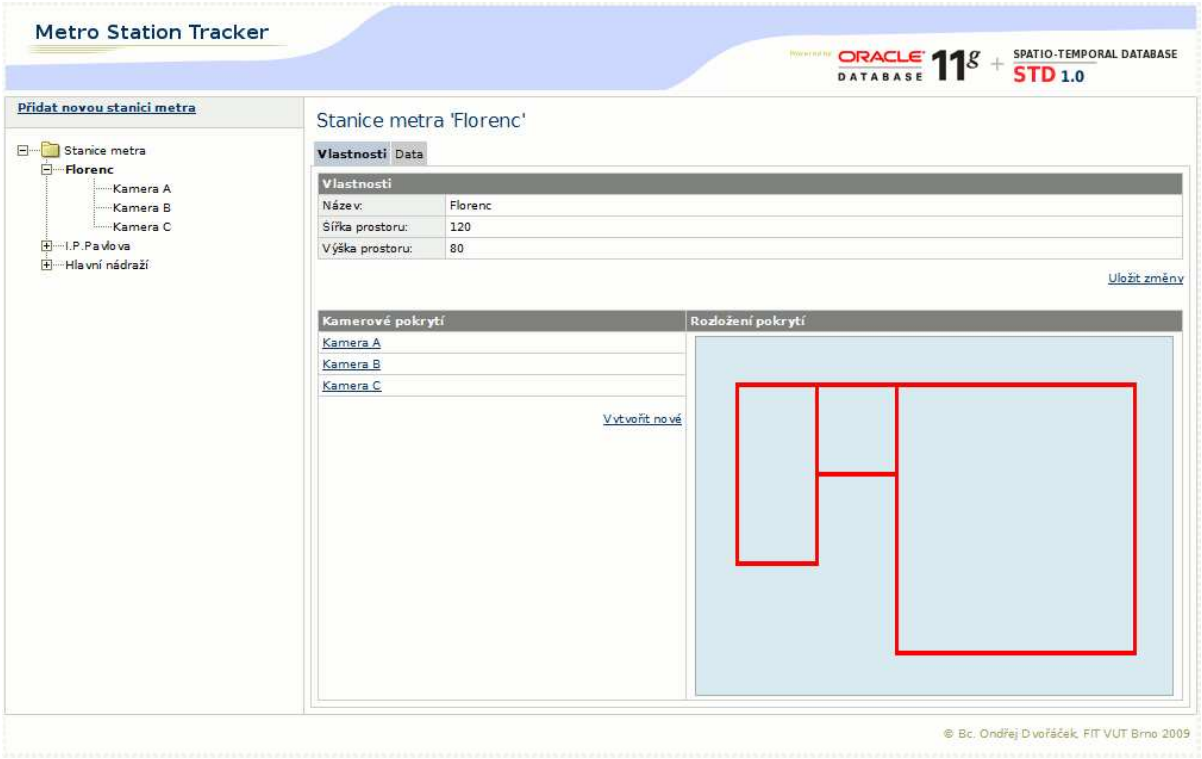
Poslední množinou objektů jsou pohybující se objekty, jejichž data byly získány pokročilými technikami zpracování obrazu a videa z kamer dohledového systému. Napojení na takový systém zatím neexistuje, proto jsou data importována do databáze přímo hromadně pomocí SQL příkazů.



Obrázek 6.1: Konceptuální diagram tříd aplikace Metro Station Tracker.

6.1.2 Správa stanic metra

Stanice metra, které tvoří základní strukturní prvek ukázkové aplikace, je možné vytvářet, upravovat, či mazat. Pro zobrazení informací o konkrétní stanici metra slouží její detail zobrazený vždy po kliknutí na odkaz na některou z definovaných stanic metra (náhled grafického uživatelského rozhraní viz obrázek 6.2). Na tomto místě má potom uživatel také možnost změnit základní parametry stanice, jako její název, či rozměry plochy, nebo přejít k detailům jednotlivých kamerových pokrytí, či zobrazení stránky pro sledování pohybujících se objektů odpovídajících zobrazované stanici metra.



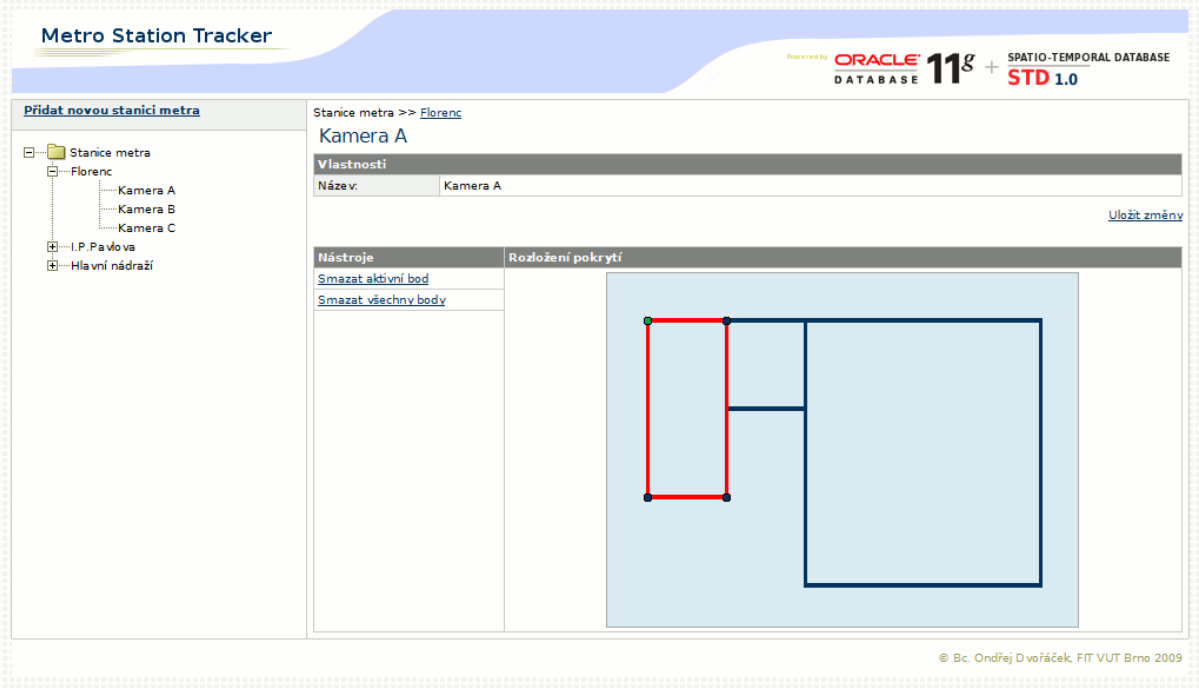
Obrázek 6.2: Grafické uživatelské rozhraní správy stanice metra.

6.1.3 Správa kamerového pokrytí

Pro každou stanici metra je možné přidávat libovolné množství kamerových pokrytí, avšak vždy alespoň jedno. Tyto definované regiony snímání je možné označit svým názvem a v případě potřeby interaktivně, pomocí speciálního ovládacího prvku grafického uživatelského rozhraní, měnit jeho prostorové rozložení (grafické uživatelské rozložení viz obrázek 6.3). Vytvoření nového kamerového pokrytí je realizováno stejným uživatelským rozhraním jako v případě jeho editace.

Prostorové rozložení regionů snímání, jak už bylo zmíněno, je zadáváno pomocí speciálního ovládacího prvku uživatelského rozhraní, který odpovídá jednoduché mapě prostoru stanice metra. Jednotlivé body vymezující tento region jsou přidávány do mapy kliknutím myši na požadovanou pozici. Body jsou následně označeny ikonkami, které se liší podle toho, zda se jedná o běžný bod

nebo o aktivní bod. Každý z běžných bodů je možné aktivovat opětovným kliknutím na jeho ikonku na mapě. Jednotlivé body je pak také možné tažením myši různě po mapě přesouvat. Aktivní body je navíc možné smazat. Všechny zadané body je také možné vymazat jediným stiskem tlačítka, které ovšem žádá potvrzení tohoto úkonu. Pro uložení definovaného mapového pokrytí je nutné zadání alespoň tří takových bodů.

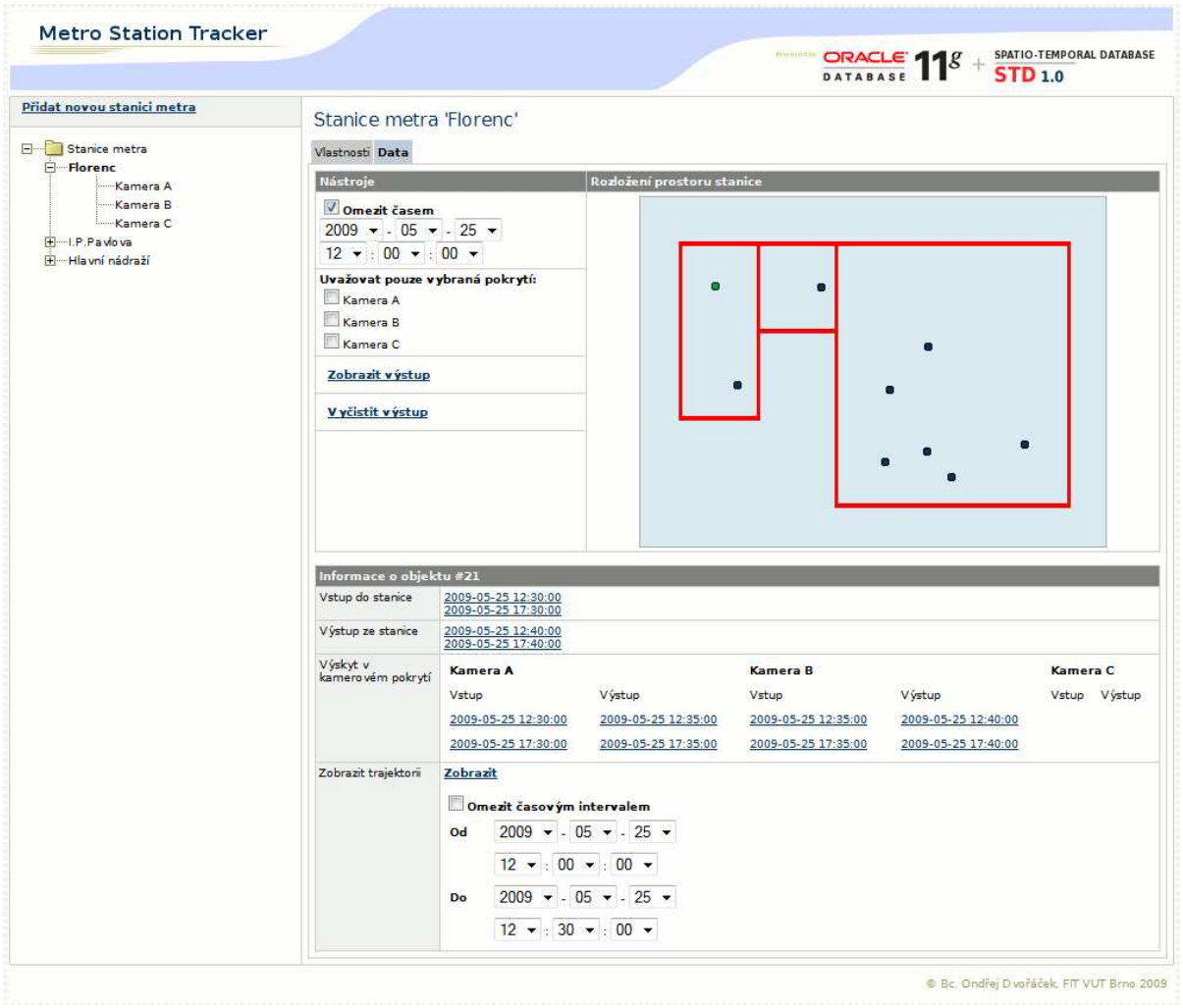


Obrázek 6.3: Grafické uživatelské rozhraní správy kamerového pokrytí.

6.1.4 Sledování pohybujících se objektů ve stanicích metra

Nejdůležitější funkcionalitou této ukázkové aplikace je možnost sledování pohybujících se objektů v rámci vybrané stanice metra (grafické uživatelské rozhraní viz obrázek 6.4). Jednotlivé pohybující se objekty jsou zobrazeny do podobného ovládacího prvku jako v případě zadávání kamerového pokrytí a pomocí doplňkových nástrojů je možné tyto objekty filtrovat podle časového okamžiku, nebo podle prostoru vybraných kamerových pokrytí. Výstupem takového zobrazení je množina bodů rozmístěných po mapě umožňujících po kliknutí zobrazení bližších informací o pohybujícím se objektu.

Informace zobrazené o vybraném objektu obsahují časy jeho vstupů do a výstupů z, jednak samotného prostoru stanice metra, a dále pak prostorů jednotlivých kamerových pokrytí. Aplikace navíc umožňuje zobrazení tohoto vstupního či výstupního místa na mapě po kliknutí na vybraný časový údaj o vstupu či výstupu do respektive z vybraného prostoru. Pro získání představy o pohybu vybraného objektu v prostoru stanice metra je navíc doplněna funkce pro zobrazení celkové trajektorie pohybu bodu nebo trajektorie omezené zadaným časovým intervalem.



Obrázek 6.4: Grafické uživatelské rozhraní pro zobrazení dat pohybujících se objektů.

Implementace jednotlivých funkcionalit nabízených tímto rozhraním je provedena za pomocí podpory STD využité jak k implementaci složitějších databázových funkcí jako je například:

- `FUNCTION camspace_entry(mp_id NUMBER, camsp_id NUMBER) RETURN STD_TPOINT_TYPE_ARRAY`

  Získá kolekci temporálních bodů odpovídajících vstupům do zadaného kamerového pokrytí. Funkce kombinuje použití funkcí *at_region* (viz kapitola 5.4.3) a *get_initial_tpoint* (viz kapitola 5.4.2).

nebo ke skládání složitějších SQL dotazů, například:

- Výběr pohybujících se bodů procházejících regionem:

```
SELECT mp.mpoint_id, mp.get_intial_tpoint() AS tp
FROM mst_moving_points mp, mst_camera_spaces cs
WHERE
  cs.id = `21` AND
  STD_TEMPORAL.passes( VALUE(mp), VALUE(cs) ) = `TRUE`
```

6.2 Ovládání aplikace

Ovládání aplikace je dáno svým charakterem, a to tím, že se jedná o webovou aplikaci. Jednotlivé ovládací prvky jsou tak běžné a ověřené dlouholetým používáním internetu skupinami uživatelů s různými zkušenostmi při práci s osobními počítači. Orientace mezi jednotlivými stránkami aplikace je intuitivní a doplněna o jednoduché krátké nápovědy nebo vysvětlující popisky tlačítek. Na každé stránce je navíc odkaz umožňující přechod zpět na tu stránku, která je v kontextu zobrazovaného obsahu stránky o úroveň výše (například z detailu kamerového pokrytí zpět ke stanici metra, ke které daný region snímání náleží).

Výchozím stavem aplikace je úvodní stránka se základními informacemi o aplikaci. Na této stránce jsou uživateli také nabídnuty základní úkony, které mohou vést ke snadnějším začátkům práce s touto aplikací.

Mezi definovanými stanicemi metra je možné se orientovat pomocí menu v levé části stránky. Názvy jednotlivých stanic spolu s odkazy na zobrazení jejich detailu jsou organizovány do podoby rozbalovacího stromu s podkategoriemi odpovídajícími odkazům na definované kamerové pokrytí každé z vybraných stanic metra.

Pro zobrazování prostorových informací je použit speciální prvek grafického uživatelského rozhraní umožňující posouvání a přibližování aktuálního výřezu zobrazované části prostoru stanice metra. Díky tomuto prvku je také možné zadávat rozložení nových kamerových pokrytí a také interaktivně vybírat zobrazené pohybující se objekty.

Ostatní doplňkové informace týkající se zejména instalace a uvedení do provozu této aplikace jsou uvedeny v uživatelské příručce, která je součástí přílohy 1 této diplomové práce.

7 Zhodnocení dosažených výsledků

Jedním z nejdůležitějších přínosů této diplomové práce se stalo nastudování základní formální specifikace systému typů a operací časoprostorových dat zavedené v práci Ralfa Hartmuta Gütinga [1] a na základě tohoto studia potom vytvoření databázového rozšíření STD umožňujícího uložení a přímé dotazování časoprostorových dat.

V následujících podkapitolách je jednak STD porovnáno s původní databází pohybujících se objektů, ze které měla tato práce vycházet, a dále jsou zde uvedeny náměty na jeho další vývoj, tak aby bylo docíleno kvalitní, robustní a přitom jednoduché podpory pro práci s časoprostorovými daty v databázovém systému Oracle.

7.1 Porovnání STD s původní databází pohybujících se objektů

Podrobným studiem databáze pohybujících se objektů, která měla být základem pro tuto diplomovou práci, se objevilo několik zásadních důvodů pro vytvoření zcela nové databázové podpory pro reprezentaci časoprostorových dat s možností se nad takovými daty přímo dotazovat.

Obě řešení jsou navržena a implementována v prostředí databázového systému Oracle, přičemž původní databáze pohybujících se objektů je navržena pro verzi tohoto systému 10g a STD využívá novější verzi 11g. V souvislosti s řešenou problematikou však mezi těmito verzemi není žádný zásadní rozdíl. V obou případech je také využito možnosti definice uživatelsky definovaných objektových datových typů.

Zatímco původní databáze pohybujících se objektů, zkráceně DPO, využívá možnosti zapouzdření operací k definovaným datovým typům pouze pro statické metody, které slouží většinou jako seskupené binární operátory konkrétního datového typu, jsou v STD k jednotlivým definovaným datovým typům přidány pouze ty metody, které přímo manipulují konkrétní instanci datového typu. Základní operace STD, které ve skutečnosti odpovídají převážně binárním operátorům, jsou pak podle domény svého použití seskupeny v doplňujících databázových balíčcích. Tyto balíčky jsou navíc doplněny o základní konstantní proměnné společné pro všechny definované operace (např. hodnota výchozí prostorové tolerance).

Dalším významným rozdílem je způsob práce se samotnými časoprostorovými daty. DPO pro manipulaci těchto dat vyžaduje vytvoření svých vlastních databázových tabulek a s těmi potom pracuje ve svých časoprostorových operacích. Tento fakt však zamezuje jakémukoliv dalšímu využití DPO pro specifické implementace. STD proto pracuje pouze s daty objektů, které buď mohou existovat volně a být tak použity v blocích PL/SQL programů, nebo být uloženy v databázových

tabulkách a následně zpracovány dostupnou sadou časoprostorových operací přímo v SQL dotazech. Významnou výhodou tohoto způsobu reprezentace dat je také možnost definované typy STD jednoduše zdědit do jiných uživatelsky definovaných datových typů a doplnit je tak o potřebné atributy či operace a neztratit tak podporu pro jejich časoprostorové dotazování, podobně jako tomu je v ukázkové aplikaci popsané v kapitole 6.

V souvislosti s uložením dat se obě řešení liší i v pohledu na indexaci časoprostorových dat. V DPO jsou data uloženy v databázových tabulkách, a proto je možné vytvořit nad prostorovou částí dat prostorový index, který je nabízen databázovým rozšířením Oracle Spatial. Jak ale bylo popsáno v kapitole 5.6, je použití tohoto indexu pro časoprostorová data nepříliš vhodné z důvodu možnosti častého vkládání nových dat (v některých případech se dá předpokládat i ukládání dat v reálném čase), a tím k časté aktualizaci samotného prostorového indexu. U STD proto zatím není databázových indexů využito, nicméně vytvoření speciálního časoprostorového indexu (námět na takový databázový index je uveden v kapitole 5.6) je jedním z hlavních cílů dalšího vývoje v oblasti dotazování časoprostorových dat.

Kromě rozdílů v architektuře se navržená řešení liší i v nabízených operacích pro dotazování časoprostorových dat. Zatímco STD nabízí komplexní souhrn základních operací, DPO je zaměřeno na výběr několika málo specifických metod, které je však možné bez problému vyjádřit základními operacemi STD přímo v dotazu SQL, např.:

- funkci `mpoints_count_region_interval` pro zjištění počtu pohybujících se bodů, které v zadaném časovém intervalu prošli regionem se zadaným ID, je možné vyjádřit jako:

```
SELECT count(mp.id)
FROM mpoints mp, regions r
WHERE
   r.id = '1' AND
   STD_TEMPORAL.passes(
     STD_TEMPORAL.at_periods(
        VALUE(mp),
        NEW STD_PERIODS_TYPE(
          TO_DATE('2009-05-20 12:30:00'), 'FALSE',
          TO_DATE('2009-05-20 13:00:00'), 'FALSE'
        )
     ),
     VALUE(r)
   ) = 'TRUE'
```

7.2 Náměty na další vývoj STD

Jelikož tato diplomová práce dala příležitost k nastudování v celku rozsáhlé problematiky reprezentace a dotazování časoprostorových dat, bude hlavním cílem pokračování této práce kompletní zavedení formální specifikace systému datových typů a operací spojených s časoprostorovými daty a tedy i daty pohybujících se objektů.

Společně se zaváděním výše zmíněného formálního systému bude nutné vytvořit speciální, v této práci již několikrát diskutovaný, časoprostorový index, přičemž v obou případech by se mělo vycházet z praktických zkušeností získaných použitím základní implementace rozšíření STD, které je hlavním výsledkem této diplomové práce.

Finálním krokem řešení dané skupiny problémů spojených s časoprostorovými daty by pak měla být kompletní implementace časoprostorového databázového rozšíření pro systém Oracle, které by bylo možné zařadit na úroveň podobných rozšíření, jako je například samotný Oracle Spatial, který byl využit při implementaci STD.

Kromě výše uvedených, poměrně rozsáhlých, témat dalšího pokračování v oblasti časoprostorových dat se však nabízí i několik dílčích cílů, a to zejména doimplementování podpory STD pro ostatní temporální datové typy, jako je například *temporální* či *pohybující se region*, nebo datové typy pro v čase se měnící hodnoty základních datových typů, jako *real*, *int* či *bool*.

8 Závěr

V průběhu tvorby této diplomové práce bylo možné se seznámit s řadou specifických problémů, které doména časoprostorových dat skýtá. Pečlivě byly nastudovány především dva základní dokumenty, ze kterých měla tato diplomová práce vycházet, a to diplomová práce pana Ing. Jaroslava Vališe [2] a odborný článek pana Ralfa Hartmuta Gütinga [1]. Na základě získaného poznání pak bylo vytvořeno zcela nové databázové rozšíření, pro zjednodušení nazvané STD, které se při své realizaci opírá o formální specifikaci časoprostorových dat zavedené v druhém ze zmíněných dokumentů, tedy práci Ralfa Hartmuta Gütinga [1].

Implementovaným řešením, jehož funkčnost byla také ověřena použitím v jednoduché ukázkové aplikaci, se stala základem pro další pokračování řešení této problematiky při zpracování tématu mého doktorského studia nazvané „Databáze pohybujících se objektů“.

Směry, kterými by se další vývoj v oblasti časoprostorových dat, potažmo dat pohybujících se objektů, měl ubírat, jsou odvozeny od problémů vzniklých při tvorbě této diplomové práce a jsou uvedeny v kapitole 7.

Literatura

[1] Güting, R. H. et al: *A Foundation for Representing and Querying Moving Objects*. ACM Transactions on Database Systems. Vol. 25, No. 1 March 2000, pp. 1 – 42.

[2] Vališ, J.: *Databáze pohybujících se objektů*. Diplomová práce. Brno, FIT VUT v Brně, 2008.

[3] Vetešník, J.: *Indexování pohybujících se objektů*. Diplomová práce. Brno, FIT VUT v Brně, 2008.

[4] *Oracle Spatial User's Guide and Reference, 10g Release 2 (10.2)* [online]. Poslední modifikace: 16. června 2005 [cit. 2009-01-07]. Dostupné z: <http://download.oracle.com/docs/html/B14255_01/toc.htm>.

[5] *Oracle Database SQL Reference, 10g Release 2 (10.2)* [online]. Poslední modifikace: 29. listopadu 2008 [cit. 2009-01-07]. Dostupné z: <http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/toc.htm>.

[6] Carvalho, A., Ribeiro, C., Sousa, A. A.: *A Spatio-temporal Database System Based on TimeDB and Oracle Spatial*. IFIP International Federation for Information Processing. Vol. 205. Boston: Springer, 2006, pp. 11 – 20. ISBN 978-0-387-34345-7.

[7] Křížová, M.: *Indexování pohybujících se objektů.* Diplomová práce. Brno, FIT VUT v Brně, 2009.

[8] *Introduction to Oracle Objects* [online]. Poslední modifikace: 23. dubna 2009 [cit. 2009-05-25]. Dostupné z: <http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28371/adobjint.htm>.

[9] *Using PL/SQL Collections and Records* [online]. Poslední modifikace: 23. dubna 2009 [cit. 2009-05-25]. Dostupné z: <http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28370/collections.htm>.

[10] *Using PL/SQL Packages* [online]. Poslední modifikace: 23. dubna 2009 [cit. 2009-05-25]. Dostupné z: <http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28370/packages.htm>.

[11] Hall, M.: *Java: servlety a stránky JSP*. Praha: Neocortex, 2001. 585 s. ISBN 80-86330-06-0.

[12] *Java EE - Wikipedie, otevřená encyklopedie* [online]. Poslední modifikace: 24. dubna 2009 [cit. 2009-05-25]. Dostupné z: <http://cs.wikipedia.org/wiki/J2EE>.

[13] *Trail: JDBC(TM) Database Access (The Java™ Tutorials)* [online]. Poslední modifikace: 18. března 2008 [cit. 2009-05-25]. Dostupné z: <http://java.sun.com/docs/books/tutorial/jdbc/index.html>.

Seznam obrázků

Seznam příloh

Příloha 1: CD nosič s přílohami v elektronické podobě. Obsahuje:

- Elektronickou podobou textu této diplomové práce.
- Zdrojové kódy STD s uživatelskou příručkou.
- Zdrojové kódy ukázkové aplikace Metro Station Tracker s uživatelskou příručkou.